Universidad de San Carlos de Guatemala
Facultad de Ingenieria
Escuela de Ingeniería en Ciencias y Sistemas

MICROSOFT .NET
SU *FRAMEWORK* POR DENTRO Y LAS BASES PARA LA CONSTRUCCIÓN DE APLICACIONES

Carlos Arnoldo Méndez Saravia
Asesorado por: Ing. Calixto Monzón

GUATEMALA, MARZO DE 2004

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

MICROSOFT .NET
SU *FRAMEWORK* POR DENTRO Y LAS BASES PARA LA CONSTRUCCIÓN DE APLICACIONES

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA

FACULTAD DE INGENIERÍA

POR

CARLOS ARNOLDO MÉNDEZ SARAVIA

ASESORADO POR: ING. CALIXTO MONZON

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, MARZO DE 2004

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Sydney Alexander Samuels Milson
VOCAL I	Ing. Murphy Olympo Paiz Recinos
VOCAL II	Lic. Amahán Sánchez Alvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Sydney Alexander Samuels Milson
EXAMINADOR	Ing. Marlon Antonio Perez Turk
EXAMINADOR	Ing. Edgar Rene Ornelyz Hoil
EXAMINADOR	Ing. Luis Alberto Vetorazzi España
SECRETARIO	Ing. Pedro Antonio Aguilar Polanco

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

MICROSOFT .NET
SU *FRAMEWORK* POR DENTRO Y LAS BASES PARA LA CONSTRUCCIÓN DE APLICACIONES

Tema que me fuera asignado por la Coordinación de la Carrera de Ingeniería en Ciencias y Sistemas, con fecha 10 de febrero de 2003.

Carlos Arnoldo Méndez Saravia

Guatemala, 27 de octubre de 2003

Ingeniero
Carlos Azurdia
Coordinador de Privados y Revisión de Tesis
Escuela de Ciencias y Sistemas

Estimado Ingeniero:

Por medio de la presente, me permito informarle que he asesorado el trabajo de graduación titulado: **MICROSOFT .NET SU *FRAMEWORK* POR DENTRO Y LAS BASES PARA LA CONSTRUCCIÓN DE APLICACIONES**, elaborado por el estudiante Carlos Arnoldo Méndez Saravia, y a mi juicio el mismo cumple con los objetivos propuestos para su desarrollo.

Agradeciéndole de antemano la atención que le preste a la presente, me suscribo de usted.

Atentamente,

Ing. Calixto Monzón
Asesor

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas

Guatemala, 15 de marzo de 2004

Ingeniero
Luis Alberto Vettorazzi España
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Vettorazzi:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **Carlos Arnoldo Méndez Saravia,** titulado: **MICROSOFT .NET SU *FRAMEWORK* POR DENTRO Y LAS BASES PARA LA CONSTRUCCIÓN DE APLICACIONES**, y que a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

Universidad de San Carlos de Guatemala
Facultad de Ingeniería

El Director de la Carrera de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor de tesis y del Licenciado en Letras, al trabajo de graduación titulado **MICROSOFT .NET SU *FRAMEWORK* POR DENTRO Y LAS BASES PARA LA CONSTRUCCIÓN DE APLICACIONES**, presentado por el estudiante **Carlos Arnoldo Méndez Saravia**, aprueba el presente trabajo y solicita la autorización del mismo.

ID Y ENSEÑAD A TODOS

Ing. Luis Alberto Vettorazzi España
Director
Ingeniería en Ciencias y Sistemas

Guatemala, 18 de marzo de 2004

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **MICROSOFT .NET SU *FRAMEWORK* POR DENTRO Y LAS BASES PARA LA CONSTRUCCIÓN DE APLICACIONES,,** presentado por el estudiante universitario **Carlos Arnoldo Méndez Saravia**, procede a la autorización para la impresión del mismo.

IMPRÍMASE:

Ing. Sydney Alexander Samuels
DECANO

Guatemala, 18 de marzo de 2004

AGRADECIMIENTOS

A Dios

Por haberme dado la oportunidad de la vida y de hacer lo que deseo.

A la Virgen de Concepción y de Guadalupe

Por permitirme cumplir mis objetivos y promesas, y darme la oportunidad de continuar adelante con mis metas.

A mis padres

Arnoldo Méndez y Marina de Méndez, por su guía, enseñanzas, ejemplo y su comprensión y apoyo incondicional.

A mis abuelos

Por toda su comprensión, apoyo y ejemplo. En especial a mi abuelo Carlos Saravia por su ejemplo de rectitud y ética.

A mis hermanas

Marina y Astrid, por su comprensión, ejemplo y apoyo en todo momento.

A mis amigos y compañeros

Por permitirme compartir esta experiencia de vida, y hacerla mejor. En especial a José Luis, Marcelo, Daniel, Otto y Sergio.

A Ing. Calixto Monzón e Ing. Guillermo Castañeda

Por su confianza y apoyo profesional, así como su apoyo para llevar a buen termino este documento.

Y a la Universidad de San Carlos de Guatemala.

DEDICATORIA

A mis padres

Por haberme dado su apoyo incondicional y permitirme tener mis propias metas y sueños.

A mis abuelos

Por su cariño y por darme la razón de superación en esta vida.

A mi familia

Por su apoyo y confianza en todo momento.

A mis compañeros

Por permitirme acompañarlos en el camino a construir una carrera profesional, y siempre brindarme su apoyo.

A las empresas ICON, TEXDISA y EEGSA

Por permitirme formar parte de ellas, en especial a Hugo Cruz, Jorge Maldonado, Byron Albanés, Alvaro Van Houtven y Guillermo Castañeda, por haberme brindado su confianza profesional para poderme desarrollar en estas empresas.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

RESUMEN

El presente trabajo de graduación es en su mayoría de tipo investigativo, aquí se incluirán las bases fundamentales del conjunto de elementos que forman la visión de desarrollo a través de un lenguaje común que se integra de forma dinámica con grupos de librerías que permiten apertura en el desarrollo de aplicaciones basadas en *Web*, cuya orientación general es enfocarse al comercio por Internet.

Entre los temas principales que se exponen, se describe en forma aplicativa los componentes del .Net *Framework*, analizando el interior de la forma en que se comporta el ensamblador de dicha tecnología, en el manejo de las estructuras y la gramática, para ello se hace una estructuración de temas relacionados en base a la pila de funcionamiento de la plataforma .Net, describiendo el interior del *Framework*, su componentes como el lenguaje común de ejecución, la biblioteca de clases, así como el manejo e interpretación del Lenguaje Intermedio del .Net y su estructuración sintáctica.

Para fortalecer los fundamentos expuestos a través del presente documento, se realizaron comparaciones con versiones anteriores de sistemas que permitan el desarrollo de aplicaciones que realizarán funciones similares a las de operaciones con objetos .Net, tal como es el caso de los objetos COM y sus generaciones, ya que esto permite tener un mejor enfoque respecto a las ventajas y desventajas de la nueva plataforma, así como su interacción con su entorno, este enfoque se basa es ubicar ventajas del desarrollo de aplicaciones con esta plataforma, con la base de utilización del XML como lenguaje universal de intercambio de información, a través del protocolo SOAP.

OBJETIVOS

General

Dar a conocer detalles de los diversos elementos que forman parte de la nueva tecnología de desarrollo Microsoft .Net, que según lo planteado por Microsoft es una herramienta capaz de adaptar las tecnologías de desarrollo pasadas hacia una nueva era de construcción de soluciones, dando una visión general de las capacidades de dicha herramienta, como un esquema de trabajo, que se ha empezado a introducir como una novedad, pero que su objetivo es establecerse como una nueva filosofía de construcción de soluciones aplicativas. Se pretende dar a conocer la información general y del interior de cada uno de los elementos de esta nueva tecnología, tal como lo es el manejo de su compilador a través del ensamblador de lenguaje intermedio, así como ejemplificar la forma en que se pueden explotar las propiedades principales que brinda el uso de esta tecnología para la creación de aplicaciones de alta calidad.

Específicos

1. Dar a conocer los fundamentos generales que constituyen el desarrollo de aplicaciones bajo el esquema .Net.

2. Dar a conocer los puntos importantes en esta tecnología, así como los puntos importantes de la nueva filosofía que conlleva.

3. Presentar el interior de las herramientas de compilación y el control de estructuras de las mismas, ya que esto permitirá la optimización en desarrollo de aplicaciones.

4. Comprensión de la metodología de desarrollo de una tecnología orientada totalmente a la globalización de las aplicaciones a través del Internet.

5. Presentar las diversas formas de desarrollo que plantea este nuevo enfoque de desarrollo, conjuntamente con su manejo a bajo nivel, que permita visualizar la carga de ejecución en el lado del cliente, y en el de los servidores.

6. Presentar las ventajas de desarrollo de aplicaciones a través de una herramienta de nueva generación que permite la construcción de aplicaciones en forma eficiente y más abierta.

7. Ejemplificar el desarrollo de aplicaciones en el frente más poderoso de esta herramienta, las formas para ASP.Net y los Servicios *Web*.

INTRODUCCIÓN

Microsoft ha expuesto que actualmente ya era necesaria una nueva plataforma, la cual fuera capaz de tomar como su eje al Internet y el uso de sus dispositivos y servicios, surgiendo esta idea por un largo tiempo, pero lo difícil era colocar todas las piezas juntas para realmente dar un funcionamiento en su mayoría exitoso. Asimismo el enfoque de .Net es ser la nueva generación de plataformas de desarrollo, visualizando esta como una generación de servidores que pueden trabajar en forma conjunta para proveer servicios que puedan correr dentro de corporaciones, y dentro de un ASP o programas creados para correr por si mismos con la capacidad de prestar un servicio específico al usuario.

El .Net *Framework* o estructura de trabajo del .Net es una nueva plataforma diseñada para desarrollo simplificado principalmente en el entorno del Internet. A nivel general, esta nueva plataforma está constituida por dos componentes bases, El lenguaje común para tiempo de corrida (*Common Language Runtime* CLR) y Las Bibliotecas de Clases del .Net *Framework* (*Class Library* CL).

El CLR es el fundamento del .Net *Framework*, y aunque se puede pensar que este es un agente que maneja código en tiempo de ejecución, este provee un núcleo de servicios tales como el manejo de memoria, administración de hileras, conexiones remotas, así como manejo de una estricta seguridad y precisión del código a ejecución, debido a que permite el manejo de verificaciones de código y compilación.

El CLR también cuenta con un robusto código para implementar una estricta verificación de tipos y la infraestructura del código el cual es llamado Sistemas de Tipos Comunes (*Common Type System* CTS). Asimismo esto asegura que todo el código sea manejado por la misma traza. Estos medios que manejan código que puede consumir otros manejos de clases, tipos y objetos mientras fuerzan estrictamente la fidelidad de tipos y de la seguridad de los mismos.

La CL es una gran colección de objetos de clases reutilizables construidas sobre las bases de la orientación a objetos, las cuales pueden ser utilizadas para el desarrollo de aplicaciones, ya sea, desde un sistema de líneas de comandos o a través de aplicaciones con un interfaz gráfico para usuario (*Graphical User Interface* GUI), para aplicaciones basadas sobre la última innovación provista por ASP.NET y los servicios *web*.

Las bibliotecas de clases del .Net *Framework*, reduce la curva de aprendizaje asociada con el uso de una pieza de código, ya que facilita el desarrollo, además los componentes pueden integrarse fácilmente con las clases definidas en el .Net *Framework*. Una ventaja de las bibliotecas de clases, es que incluyen tipos que soportan una variedad de escenarios específicos de desarrollo, pudiéndose desarrollar algunos tipos de aplicaciones y servicios tales o como: consolas de aplicaciones, formas para Windows basadas en un GUI, aplicaciones basadas en ASP.NET, servicios *web* y servicios de sistema operativo *Windows*.

1. GENERALIDADES Y ORÍGENES DEL .NET

1.1 Origen de la creación del Microsoft .Net

Durante los últimos años la informática ha tenido mucho cambios, hoy en día se puede observar el valor que tiene el Internet, por lo cual, para Microsoft la creación de esta nueva filosofía de construcción de soluciones no es algo nuevo, sino es algo que le llevó alrededor de cuatro años, así como una evolución radical en el tipo de aplicaciones, esto debido al crecimiento tan abrupto del Internet, y de las puertas que este tipo de comunicación abriría, principalmente, cuando dejó de ser únicamente un medio de publicación de información y empezó a colocarse como un medio de comunicación interactiva, naciendo el "*Business to Business*" y el "*e-commerce*", que en la actualidad son muy comunes, surgiendo cada vez más las aplicaciones *Web*, siendo así el Internet el medio más fácil para la generación de negocios orientados a un mundo globalizado, ya que permiten la negociación a través de portales electrónicos sin necesidad de contar con representaciones de ventas en cada área donde se desea negociar, siendo de gran beneficio para la inversión de las empresas, sin embargo, las condiciones de heterogeneidad de los sistemas hacía muy compleja la interacción de diversas plataformas y sistemas, surgiendo de aquí la idea de Microsoft, que va más allá de la creación de una nueva herramienta de desarrollo, sino de los conceptos necesario para formar y facilitar un tipo de desarrollo estandarizado, capaz de no depender de plataformas específicas para su desarrollo, lo cual aplica desde el sistema operativo, que se retome el espíritu inicial de la comunicación común del Internet, y para la realización de esto, se buscó la utilización del XML (*Extensible Markup Language* o Lenguaje de Marcado Extensible).

Además de la busqueda de independencia de plataformas para el desarrollo, era necesario poder utilizar herramientas más poderosas para el desarrollo de aplicaciones similares a las establecidas en los sistemas cliente-servidor, y que fueran capaces de prestar un nivel de funcionamiento lo suficientemente optimo como para hacer que su uso fuera atractivo, de allí el surgimiento de un interprete de datos que es la base más firme de la creación del .Net, fué nombrado ***Common Language Runtime***, y permite interpretar un lenguaje intermedio al cual fueron compiladas las aplicaciones realizadas en cualquier lenguaje, ensamblándose de una forma común que permita ejecutar aplicaciones en los sistemas operativos como Windows, sin necesidad de hacer procesos de registro de librerías que en muchas ocasiones provocaban problemas de compatibilidad entre aplicaciones, así como el hacer uso de información de diversos orígenes (incluso fuera de la empresa).

1.1.1 El problema de la infraestructura común

Actualmente el hardware y el ancho de banda de las telecomunicaciones han reducido sus costos en forma considerable, sin embargo, aún son un tropiezo para la apertura en el manejo de información, aunque la desaparición de este obstáculo es cada vez mayor, debido a que el hardware ahora es mucho más fácil de acceder incluso en países cuya evolución tecnológica aún es pobre, pero además de este problema, existe otro poco visible, pero mucho más costoso, como lo es la utilización de software, tal como se pueden observar en la actualidad las aplicaciones para Internet no están formando parte de las lógicas del negocio, sin embargo, el factor que hace que se implementen diversas aplicaciones en diferentes centros para ser utilizados, a esto Internet introduce un nuevo problema, debido al hecho de ser público, incontrolable y además de naturaleza heterogénea, ya que esto hace fácil chocar con preguntas acerca del manejo de la seguridad.

Para el funcionamiento de muchos servicios en las empresas, es necesario la utilización de servicios de terceros, los cuales pueden ser automatizados al consultar cierta información del proveedor, sin embargo por las mismas condiciones de seguridad es difícil que las empresas liberen su información, asimismo las condiciones de heterogeneidad de los sistemas lo hace aún más difícil, de aquí surge la necesidad planteada por Microsoft de una infraestructura común a nivel de *Software*, de igual forma que la existente en la comunicación, para poder realizar de esta forma transmisión de información en forma estandarizada que permita una interacción de sistemas sin arriesgar la seguridad del control de información naciendo así lo que Microsoft a denominado ***e-services***.

Parte muy importante de esta infraestructura común es la presencia de estándares para el intercambio de información como lo es el XML, de igual forma que la existencia de herramientas que lleven a un resultado común, para el desarrollo de aplicaciones cada vez más complejas, de aquí que la visión de Microsoft de vender servicios en un futuro próximo, en el cual las compañías ofertarán servicios usuarios interesados podrán consumirlos, de igual forma muchos de estos servicios serán gratuitos o simplemente intercambiados entre diversas empresas, o vendidos a través de planes de suscripción, o intercambiados por otros servicios, pero lo importante es que estos servicios se podrán integrar a la lógica de negocios. Algunos de los servicios que plantea esta visión actualmente se encuentran en funcionamiento y muchos otros en versiones Beta:

- Validación de tarjetas de crédito.
- Dar rutas desde un punto inicial a un punto final.
- Visualización de menús de restaurantes.
- Pagar cuentas a través de cuentas de cheques.
- Rastrear paquetes de mensajería.

En este nuevo mundo, los servicios están conectados al Internet, por lo cual los desarrolladores buscan un camino fácil para acceder a estos servicios, y parte de la iniciativa del .Net es proveer esta plataforma de desarrollo, y aunque Microsoft se plantea como líder en desarrollo y en la definición de estándares involucrados en esta tecnología, ellos no son dueños de ninguno de estos estándares, de lo cual surge la metodología de comunicación en la que los equipos clientes describen a un servidor su requerimiento por XML y enviándolo a través del protocolo tgh, posteriormente el servidor sabrá como interpretar los segmentos de código XML, procesando el requerimiento y retornando la respuesta en XML al cliente, y de aquí surge el término SOAP (*Simple Object Access Protocol*) que describe el formato específico de XML cuando este se envía a través del HTTP, y que describiremos a mayor detalle más adelante. Algo que hace importante la utilización del SOAP es que podrá ser utilizado por cualquier sistema de pueda escuchar requerimientos con un puerto TCP/IP, y pueda leer y escribir sobre dicho puerto es suficiente, ya que en un futuro no muy lejano estos sistemas no serán solamente computadoras, sino también teléfonos móviles, automóviles, etc.

1.1.2 Las generaciones de herramientas de desarrollo

Durante mucho tiempo se ha producido el desarrollo de aplicaciones para computadora que permitan la optimización de procesos, así como permitan la interacción de procesos complejos surgiendo varias generaciones que han evolucionado hasta llegar a un punto de volverse servicios básicos para el funcionamiento adecuado de muchas empresas, de estas generaciones daremos una síntesis para visualizar el origen de una plataforma que pretende esquematizar una utilización de procesos complejos por medios de comunicación públicos, sin tener que duplicar entornos de funcionamiento de procesos.

Hasta 1975, fue la generación de las aplicaciones en grandes equipos y con reglas muy rígidas y complejas, de igual forma que con un enfoque bastante tosco hacia el cliente, siendo en esta época la utilización de clientes que eran únicamente consolas o terminales tontas, sin la capacidad de procesamiento, el cual se realizaba completamente en el servidor, utilizándose herramientas de desarrollo tales como ***Cobol*** y **RPG**.

De 1975 a 1990, fue la generación en que surgieron aplicaciones para PC, cuya principal diferencia fue la utilización de procesamiento local, es decir, los clientes dejaron de ser únicamente herramientas de presentación y captura de datos, sino también empezaron a realizar procesos de cálculo y almacenamiento de información, utilizándose herramientas de desarrollo como ***MS-Basic***, ***Pascal***, ***Quick Basic*** y otras.

De 1990 en adelante surge el desarrollo de aplicaciones basadas en procesamiento local, pero con el manejo de un interfaz gráficos (GUI), que con el tiempo se hizo cada vez más detallado impulsando el crecimiento de aplicaciones amigables, ya que su contenido gráfico permitía un entendimiento de la funcionalidad de los sistemas sino completamente claro por lo menos de tal forma que permitiera una interacción simple con el usuario, para este desarrollo se utilizaron herramientas que fueron llamadas Visuales tales como ***Visual Basic, Visual C++, Power Builder***, ***Delphi*** y muchas otras más. Estas aplicaciones hoy en día aún forman parte importante de la funcionalidad de las empresas, a pesar de ser una generación pasada, pero en cuanto al manejo de la lógica del negocio ha hecho que este tipo de aplicaciones hayan evolucionado en etapas según su funcionalidad para el manejo de información, tal como ha sido la utilización de aplicaciones en n capas.

De 1997 a 2001 surgió el desarrollo de aplicaciones de publicación de información, esto debido al auge del Internet, generándose el lenguaje **HTML** con lo que se inicia la utilización y la evolución de herramientas de interpretación del mismo, que fueron llamados ***Browsers***, posteriormente el código de estas aplicaciones evolucionó de tal forma que las aplicaciones no fueran totalmente de publicación de información, sino surge la utilización de los ***Scripts***, que permiten la ejecución de segmentos de lenguajes realizando procesos mucho más poderosos que los que podían hacer el simple HTML, siendo los más utilizados los ***Visual Basic Scripts*** y los ***Java Scripts***.

A partir de 2001, surgió una nueva generación de desarrollo, basada en integrar el manejo de información por medio de publicaciones dinámicas en el Internet, permitiendo la utilización de la lógica de negocios en un medio mucho más abierto, evolucionándose a una nueva combinación de fundamentos de utilización de aplicaciones, combinando el poder de las realizadas en herramientas visuales, con la versatilidad y la apertura de las aplicaciones realizadas en Internet, aumentando de esta forma la utilización de un estándar de comunicación entre sistemas como lo es el XML, naciendo así a la luz pública la utilización de herramientas capaces de hacer mucho más funcional la interpretación y el desarrollo de aplicaciones para este lenguaje, basándose como eje central en la comunicación de diversos sitios físicos a través del Internet y el protocolo HTTP, es decir el SOAP, una de las herramientas que fueron lanzadas al mercado y oficialmente en Guatemala el 11 de abril de 2002, fue el *Microsoft Visual Studio* .Net, siendo una herramienta que dentro de sí contiene herramientas que permitan el desarrollo para el Microsoft .Net, buscando la utilización de los estándares mencionados antes, haciendo aplicaciones independientes de los equipos y de los sistemas operativos, pero con capacidades mucho mayores en cuanto a ejecución de aplicaciones a través de los *scripts*.

1.2 Conceptos generales del .Net

En este segmento se incluirán algunos de los conceptos generales utilizados pro Microsoft, para su tecnología .Net, sin embargo son muchísimos los conceptos que se usan, por lo cual solamente trataremos algunos conceptos que se utilizarán posteriormente en el desarrollo de los diversos temas incluidos en el presente documento.

Dominio de aplicaciones (*application domain*). Este es un límite que el **Lenguaje Común de Ejecución** establece alrededor de los objetos creados con la misma aplicación, en otras palabras, dondequiera a lo largo de la secuencia de activación de objetos iniciando con el punto de entrada de la aplicación. Este dominio ayuda a aislar los objetos creados en una aplicación. Múltiples dominios de aplicaciones pueden existir en un proceso simple, asimismo cada uno de estos puede manejar sus propias políticas de seguridad.

Servidor de controles de ASP.NET. Un componente del lado del servidor es el que encapsula la interfaz de usuario y su funcionalidad, asimismo estos se manejan directa o indirectamente desde los controles de sistema, estos están incluidos en los controles de servicios Web, como los controles HTML, controles móviles, etc.

Controles de validación de servidor (*validation server controls*). Son controles incluidos en ASP.Net para pruebas por ingresos en HTML y de controles Web para requerimientos de programas definidos. Los controles de validación ejecutan ingresos chequeándolos en el código del servidor. Si el usuario está trabajando sin un *Browser* que soporte DHTML, los controles de validación podrán también ejecutar validaciones usando scripts de cliente.

Anfitrión de ejecución (*runtime host*). Este es el entorno de ejecución, tal como un ASP.Net, o *Internet Explorer*, o el *Shell* de *Windows*, en los cuales se puede ejecutar una aplicación y es típicamente arrancada y administrada.

Ensamblado (*assembly*). Estos son colecciones de estructura construidas funcionalmente, versionadas y distribuidas como lo puede ser una simple unidad de implementación, siendo los primeros bloques en construcción de una aplicación .Net, donde está todo el manejo de tipos y recursos los cuales son marcados uno a uno como accesibles solo con sus unidades implementadas o como exportadas para uso de código externo en dicha unidad. En el tiempo de ejecución, este ensamblado establece como poder resolver el requerimiento y los límites visibles para la aplicación, siendo forzada a ellos. Asimismo en el tiempo de ejecución se puede determinar y localizar el ensamblado para ejecutar un objeto porque todos los tipos son cargados en el contexto de este.

Cache de ensamblado (*assembly cache*). Es una vasta máquina de código en Cache usada para almacenamiento de lado a lado de ensamblados, siendo dos partes para el cache, el primero que **el cache global de ensamblado**, que contiene los ensamblados que son explícitamente instalados para ser compartidos para muchas aplicaciones en la computadora; y el segundo es el **cache de descarga** que almacena código bajado desde Internet o desde sitios de Intranet, aislando a las aplicaciones que disparan las descargas así como el código descargado para la utilización de aplicaciones para que esto no impacte a otra aplicación.

Manifiesto del ensamblado. Es una parte integral de cada ensamblado, proporciona la descripción del mismo, establece la identificación del ensamblado, y especifica los archivos que utiliza su implementación, de igual forma especifica los tipos y los recursos que utilizará, proporciona detalle de dependencias en tiempo de compilación sobre otros ensamblados, y especifica los permisos requeridos para una ejecución apropiada. Esta información es utilizada en tiempo de ejecución para resolver referencias, asimismo válida la integridad y la carga de ensamblados, y por si mismo describe la naturaleza del ensamblado.

Ensamblado compartido (*shared assembly*). Estos son ensamblados que pueden ser referenciados por más de una aplicación, es decir que son construidos específicamente para ser compartidos a través de un nombre fortalecido criptográfico.

Firma (*signature*). Esta es la lista de tipos involucrados en la definición de un tipo, campo, propiedad o variable local. Para un método, a firma incluye el nombre, número de parámetros, sus tipos, el tipo que retorna (de ser necesario), asimismo esta firma para una propiedad es similar a la de un método, y la de campos y variables locales es simplemente su tipo.

Nombre fortalecido. Este es un nombre que consiste de una identificación del ensamblado (este ID es un texto con nombre, versión una cultura de información), fortaleciéndolo con una clave pública y una firma digital generada sobre el ensamblado, basada en la información contenida en el manifiesto del mismo, por lo cual de existir varios ensamblados con el mismo nombre fortalecido, se esperaría que estos fueran idénticos.

Dominio neutral. En este dominio se realiza la carga de un ensamblado que no tiene especificado un dominio en el cual puede ejecutarse. Al cargar un ensamblado en un dominio neutral se consigue habilitar el código del ensamblado, para que sea asociado para la ejecución con sus datos y a sus estructuras, compartidas con todos los dominios de aplicaciones.

Ejecución lado a lado (*side-by-side execution*). Esta es la habilidad de ejecutar múltiples versiones de un mismo ensamblado en forma simultánea. Esto puede ser en la misma máquina o en el mismo proceso o dominio de aplicación, siendo este tipo de ejecución esencial para el robusto soporte de versiones en el CLR.

Atributo (*attribute*). Son una descripción de la declaración que anota elementos tales como tipos, campos, métodos y propiedades. Los atributos son grabados con los metadatos de un archivo de .Net *Framework*, y pueden ser usados para describir código en el tiempo de ejecución así como para afectar el comportamiento de las aplicaciones en el mismo.

Metadatos. Estos son la información que describe cada uno de los elementos manejados por el CLR, como lo puede ser un ensamblado, archivos cargados, tipos, métodos, etc. Estos pueden incluir información requerida para el debugeo y para el colector de desperdicios, así como los atributos de seguridad, clases extendidas y definición de miembros, así como cualquier otra información requerida por el CLR para la ejecución de código.

Módulo. Son unidades cargables, las cuales pueden contener declaraciones de tipos, así como la implementación de los mismos. El módulo contiene suficiente información para habilitar en el CLR la ejecución del mismo. El formato para los módulos es una extensión de los archivos ejecutables portátiles PE. Cuando se desarrolla, los módulos están siempre contenidos en un ensamblado.

Tipo de valor. Es un tipo de dato que describe completamente un valor para especificar la secuencia de bits que constituyen la representación del mismo, pero estos tipos de información para una instancia de tipo de valor no son almacenados con las instancias en el tiempo de ejecución, pero están disponibles en los metadatos, asimismo estas instancias de tipo de valor pueden ser tratadas como objetos usando el encajonamiento.

Encajonamiento (*boxing*). Es la conversión de una instancia de tipo de valor a un objeto, lo cual implica que la instancia pueda acarrear completo el tipo de información en tiempo de ejecución, y que puede estar localizada en la pila de memoria. El MSIL introduce un conjunto de instrucciones que permiten realizar esta conversión desde tipo de valor a un objeto a través de una copia del tipo de valor que se hace encajar como un nuevo objeto.

Lenguaje común de ejecución (*common language runtime* CLR). Este es el motor en el núcleo de manejo de ejecución de código. Esta ejecución provee el manejo de código con servicios tales como la integración de cruce de lenguajes, código de seguridad de acceso, administración de tiempo de vida de los objetos, de igual forma que el soporte de perfiles.

Lenguaje intermedio (Microsoft *intermediate language* MSIL). Este lenguaje intermedio es uno de los componentes más importantes del .Net *Framework*, debido a que es en cierta forma uno de los núcleos en los que se basa la filosofía del .Net, siendo este en pocas palabras un lenguaje usado como salida de diversos compiladores de lenguajes de una capa superior, asimismo es una entrada para un compilador JIT. El CLR incluye varios compiladores JIT para la conversión del código nativo del MSIL. (Debido a su importancia este tema será complementado más adelante en forma más detallada).

Código manejado (*managed code*). Este código corre bajo un convenio de cooperación con el CLR, siendo el que provee los metadatos necesarios para que el CLR provea los servicios necesarios tal como el manejo de memoria, integración de cruce de lenguajes, así como el control del tiempo de vida de los objetos. Todo código basado en el Lenguaje Intermedio de Microsoft MSIL es ejecutado como código manejado.

Código no manejado (*unmanaged code*). Ese el código que está creado sin respetar las convenciones y requerimiento del CLR, siendo este código ejecutado en el entorno del CLR con los servicios mínimos como los son el manejo de colecciones de desperdicio, así como un debugeo limitado, etc.

Cubierta de llamadas COM (c*allable wrapper* CCW). Es un objeto delegado generado por el CLR para aplicaciones COM que puede usar el manejo de clases incluyendo las clases .Net en forma transparente.

Código de seguridad de acceso. Este es un mecanismo provisto por el CLR por lo cual el manejo de código es a través de la concesión de permisos, por medio de las políticas de seguridad, los cuales son forzados, limitando que operaciones en el código se permiten ejecutar, esto para prevenir la exposición a un código de seguridad vulnerable.

Declaración de chequeo de seguridad. Es la declaración de la información en los metadatos para el control de la seguridad y es utilizado tal como declaraciones las cuales son usualmente escritas como atributos comunes, por lo cual los desarrolladores pueden invocar las bondades de la funcionalidad de la seguridad, tales como la requisición de permisos para unir referencias al código, o la requisición de permisos para recibir un tipo de dato, asimismo solicitar llamadas que tienen ciertos permisos de seguridad.

Chequeo imperativo de seguridad. Este es un chequeo de seguridad que ocurre cuando un método de seguridad es llamado con el código que está protegido. Este tipo de chequeo puede estar manejando datos y puede estar aislado a una simple localización dentro de un objeto o método. Por ejemplo si el nombre de un archivo está protegido este lo sabrá solamente en tiempo de ejecución.

Archivo ejecutable portátil (*potable executable file* PE). Un archivo en este formato es el que puede ser cargado en memoria y ejecutado por el sistema operativo cargado. Esto puede ser un archivo **.Exe** o **.Dll**. En el contexto .Net un archivo ejecutable portátil puede ser interpretado por el CLR trasladado a un código nativo el cual puede ser ejecutado por el sistema operativo.

Colección de desperdicios (*garbage collection* GC). Este es el proceso del rastreo de la transición a través de todos los puntos que fueron usados activamente por los objetos para la localización de todos los objetos que pueden ser referenciados y reutilizados en un sector de memoria. La colección de desperdicio también arregla la compactación de memoria que está en uso para reducir el espacio de trabajo necesitado en la pila.

Tiempo de vida. Este es el período de tiempo que se inicia cuando un objeto es colocado en memoria y finaliza cuando el coleccionador de desperdicios borra el objeto de la memoria. Este tiempo es una característica importante debido a que el manejo de objetos a través de deshechos permite liberar memoria en desuso, lo que antes se tenia que programar y su control se volvía tan complejo que las aplicaciones perdían su nivel de funcionamiento después de cierto tiempo de ejecución.

Expresión regular. Estas expresiones son notaciones concisas y flexibles para búsqueda y reemplazo de patrones de texto. La notación comprende dos tipos básicos de caracteres, los caracteres literales, los cuales indica el texto que existe en la cadena de análisis, y el otro tipo son los metacaracteres, los cuales indican el texto que puede variar en la cadena.

Recurso. Un recurso es un grupo de datos no ejecutables que están lógicamente distribuidos en una aplicación. Un recurso podría ser mostrado en una aplicación como un mensaje de error o como parte de la interfaz del usuario, asimismo estos pueden contener datos en los formularios, incluyendo cadenas de caracteres, imágenes y objetos persistentes.

1.3 Introducción al COM+ de Microsoft

El Modelo Componentes Objeto COM (*Component Object Model*) ha traido grandes cambios en el tipo de desarrollo, debido a su amplia utilización del modelo orientado a objetos, lo que hace más fácil la vida de los desarrolladores, según David Chappell, “el construir aplicaciones de objetos es una buena mentalidad, pero siempre los lenguajes de programación han parecido tener su propia idiosincrasia y notación de lo que son los objetos, tal como C++, *Java, Delphi, y Visual Basic*, teniendo cada uno soporte de objetos en algún modo, pero la diferencia misma puede causar confusión para los desarrolladores. Esta diferencia también crea dificultades para vendedores que quieren proveer servicios de objetos en un modo estándar a sus clientes en un mismo lenguaje”. Según lo anterior, se puede definir un modelo de objetos simples, que puede ser usado en una mezcla de lenguajes. Una vez el modelo de objetos existe, los servicios que el sistema operativo provee pueden estar completamente expuestos en un modo común, despreocupándose del lenguaje, debido a que el COM define una notación de lenguaje independiente, lo cual permite que los componentes programables puedan ser reutilizables.

1.3.1 ¿Qué es el COM+?

El COM+, es una extensión del COM clásico, conteniendo algunas características que permiten mayor versatilidad en la comunicación y utilización de los objetos, facilitando la interacción de sistemas, COM+ provee una biblioteca estándar, pero en contraste con el COM, este oculta las llamadas a esta librería debajo de funciones nativas equivalentes en los lenguajes de programación. Para cumplir con lo mencionado, el compilador utiliza las bibliotecas del COM+ en tiempo de ejecución, para que encajen llamadas a esta misma biblioteca COM+ en el código binario generado.

Una de las características importantes de estos objetos, debido a que los métodos están soportados en un número de interfaces, cada una puede ser descrita usando el lenguaje de definición de interfaces del COM (IDL). Una herramienta como el compilador MIDL, compilan un objeto IDL con una biblioteca de tipos, comúnmente almacenándolos en su propio archivo. Los clientes de este objeto pueden leer esta librería y aprender como hacer llamadas sobre los métodos de este objeto, en el COM+, los desarrolladores no necesitan definir interfaces utilizando IDL, en lugar de ellos pueden solo usar su sintaxis de lenguaje de programación para definir la interfaz del objeto. El compilador para ese lenguaje trabaja con la biblioteca COM+ para generar metadatos para un objeto, siendo estos esencialmente una gran agrupación de información de los tipos en la biblioteca. Algo más interesante de los metadatos del COM+, es que pueden ser accesados a través de interfaces genéricas de acceso a datos, definidas por *OLE Database*. Estos clientes de objetos utilizan consultas de SQL en sus propios metadatos, para buscar métodos o tipos de parámetros.

Figura 1. Manejo de los objetos con COM clásico

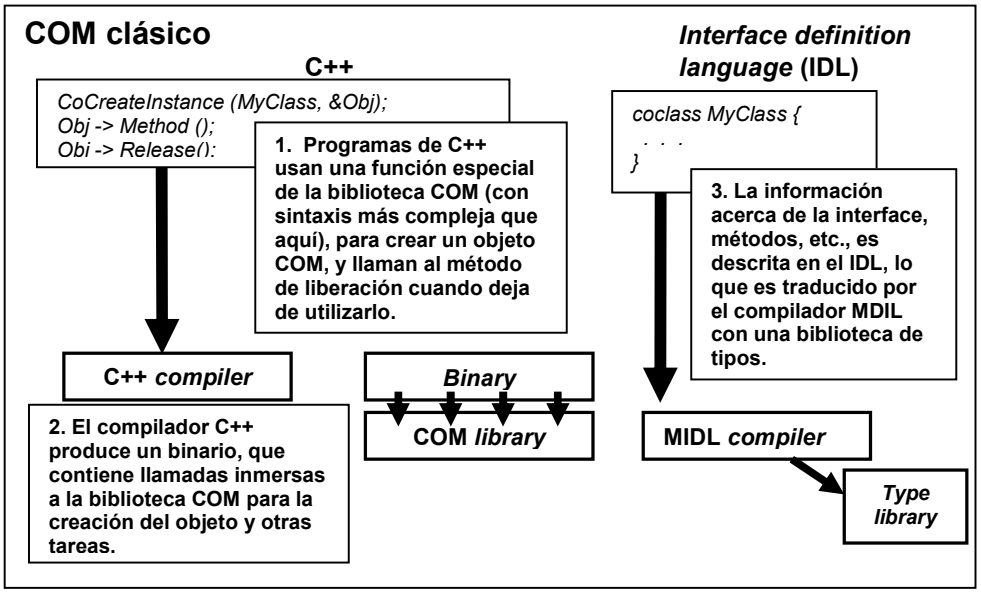
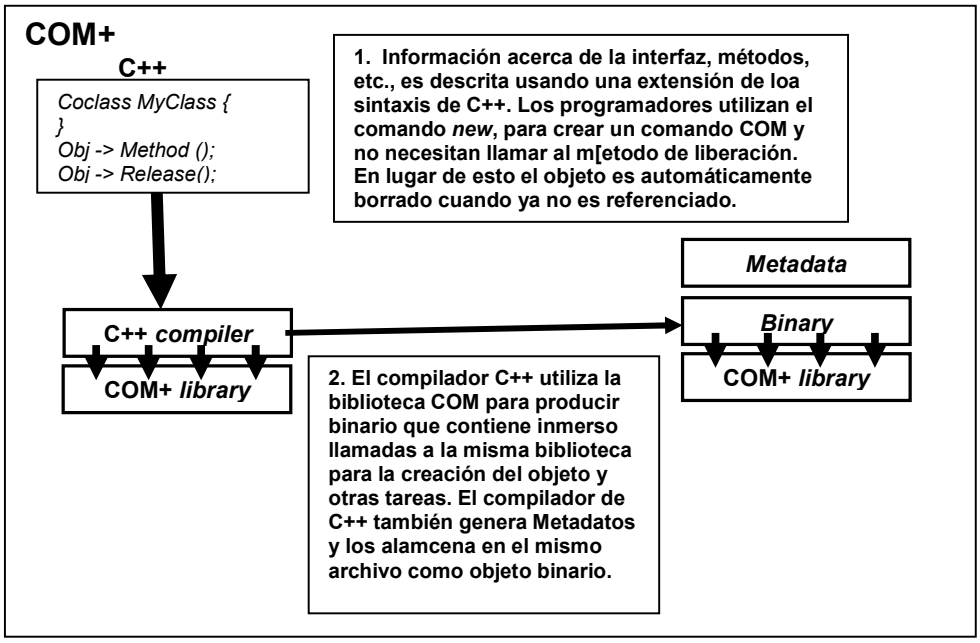


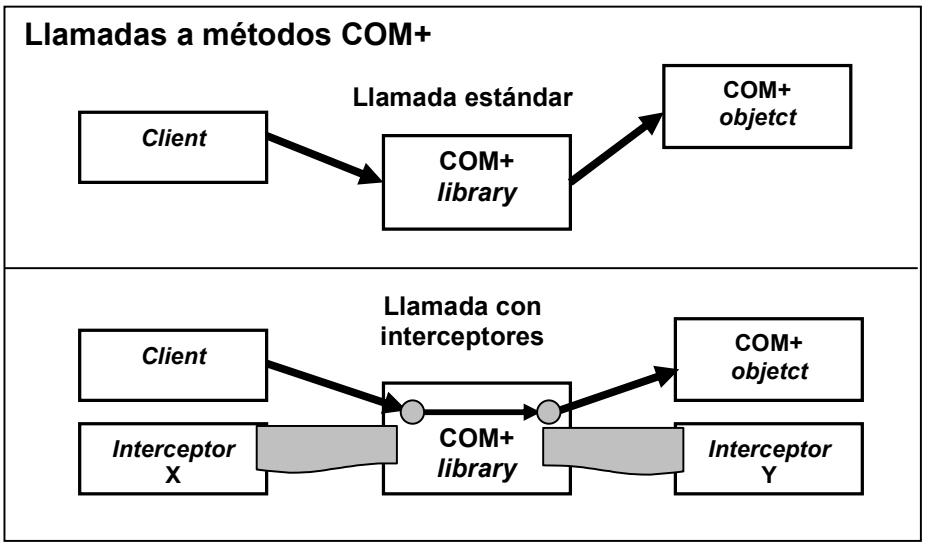
Figura 2. Manejo de los objetos con COM+



COM+ trae muchos otros cambios con respecto al COM, uno de las más importantes es que este elimina para los clientes la necesidad de ejecutar métodos para liberar objetos cuando se deja de referencia, en lugar de ello la biblioteca COM+ automáticamente toma un conteo de referencia para borrar los objetos al ya no utilizarse, evitando problemas de recarga de memoria, que siempre ha sido una de las áreas más propensas a errores. De igual forma mientras el COM siempre soportaba herencia de interfaces, el COM+ también permite la implementación de herencia entre objetos COM+ corriendo en el mismo proceso. Otra característica importante, es el soporte de constructores, que no existía en el COM, haciendo con esto a los objetos más similares a los objetos de un típico lenguaje orientado a objetos, estos constructores se ejecutan cuando un objeto es creado, permitiendo el envio de parámetros como sea necesario para el constructor, para hacer una fácil iniciación.

En el COM, la biblioteca no esta directamente involucrada con las llamadas de los clientes a un método en un objeto COM, en lugar de ello van directamente al objeto. En el COM+ toda llamada a un método COM+ pasa a través de la biblioteca COM+. El hecho de que la biblioteca esta involucrada en cualquier llamada, hace posible que se puedan insertar objetos en esa ruta. Un objeto COM+ que es automáticamente invocado durante acceso a otros objetos es llamado **Interceptor**, y uno o más de estos pueden interferir a lo largo de la ruta. Un interceptor podría ejecutar un chequeo de seguridad, esto causa que las llamadas a métodos fallarán si el permiso necesario no es otorgado, haciendo posible mantener la autenticación lógica separada desde la lógica del negocio para el componente que el cliente esta usando. Los interceptores podrían controlar la actividad del objeto para su ejecución, permitiendo construcción de aplicaciones mucho más escalables, esto es similar a que hacia el *Microsoft Transacción Server*, que ahora utiliza interceptores, y uno de ellos permitirá para cada ejecución el balance de carga, distribuyendo transparentemente las llamadas a métodos a objetos equivalentes.

Figura 3. Llamadas a métodos COM+



1.3.2 Interoperabilidad del .Net y el COM+

.Net es un revolucionario y nuevo componente de tecnología Microsoft, disponible con la liberación del *Visual Studio .Net*, que ofrece en su *Framework* controles para aplicaciones, tales como *WinForms*, ADO.Net, ASP.Net., siendo diseñado con el fundamento de simplificar el desarrollo y distribución de componentes, y al mismo tiempo pudiera proveer interoperabilidad entre lenguajes de programación en una escala sin precedente. Esto permite que no todos los códigos existentes y aplicaciones necesiten ser afectadas por .Net. De hecho esto es del todo posible para lograr desarrollar un plan de migración que pueda no solo preservar la inversión realizada por las empresas, sino también tomar las ventajas de las nuevas capacidades disponibles con .Net.

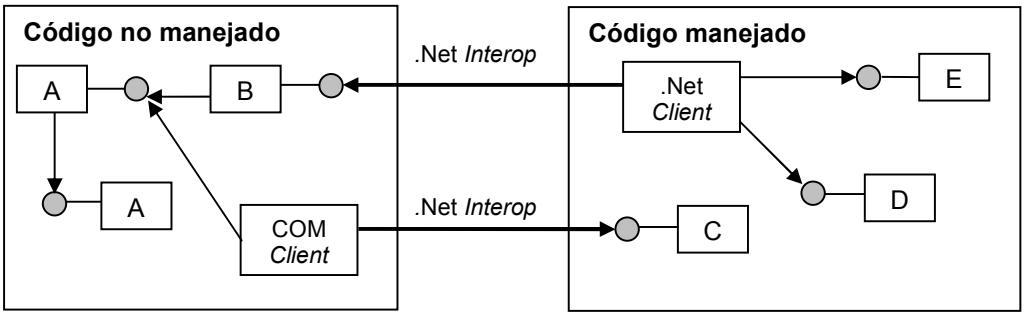
El mover código entre diferentes plataformas es conocido como ***Porting***, siendo una delicada tarea en tiempo de consumo. Muchas veces los cambios requeridos por el código existente y ejecutarse apropiadamente sobre la nueva plataforma van más allá de la sintaxis o cambios de API. Los aspectos difíciles del porting es cambiar al modelo de programación. En el caso del porting de un componente COM, una clase de C++, o un control de *Visual Basic* a .Net, estos cambios del modelo de programación van desde una determinada finalización a un colector de desperdicios, y a un manejo activo de los recursos. Un ejemplo es explícitamente el cerrar las conexiones de bases de datos, porque se puede fiar del objeto con la automática liberación de los recursos; si no se hace eso las aplicaciones no podrán comportarse de la forma en que fueron diseñadas, por lo que no podrán mantener su escalabilidad a través de los requerimientos.

La interoperabilidad es la habilidad del código para trabajar conjuntamente sobre diferentes plataformas, porque el porting es también difícil e impráctico, la interoperabilidad es lo visto, para combinarse con aplicaciones existentes COM y .Net. En .Net, cada componente COM es visto transparentemente como un componente .Net, y cada componente .Net es visualizado por el COM como un componente COM. Microsoft va fuera de su modo para asegurarse este nivel de interoperabilidad, incluyendo soporte para escenarios complejos tales como conversiones de eventos .Net a eventos COM.

Para el desarrollo de un adecuado plan de migración entre las tecnologías del COM+ y el .Net, se debe de hacer primero una clasificando de los componentes existentes, en ocasiones estos componentes podrán no ser concientes de que se está usando .Net.

Los componentes **clase A** son típicamente componentes de bajo nivel, accesados por otro componente COM, y los componentes de terceros son usados entre puntos de otra aplicación, y así sucesivamente.

Figura 4. Clases de componentes en la interoperabilidad .Net y COM+



Otra bondad de los componentes existentes, son los COM a los cuales se puede accesar usando la interoperabilidad del .Net, siendo estos los **clase B**.

De igual forma el moverse entre la parte de manejo de una aplicación, identifica un tercer tipo de componente **clase C**, que son componentes que pueden requerir proveer servicios a objetos COM, usualmente en forma de suscripción de eventos, pero posiblemente también provea servicios. Por supuesto se podrá tener componentes que el desarrollo nativamente en .Net siendo estos la **clase D**, los cuales serán la principal atracción de .Net, y ellos son probablemente con los que se pueden tomar ventajas de las capacidades de .Net, tal como los servicios *Web* y el ASP.Net, *WinForms*, etc. Finalmente se podría querer un puerto silencioso existente para .Net, es decir la **clase E**, en la cual se podría necesitar como un puerto para el núcleo de la lógica del negocio, para asegurar que el corazón de la aplicación este sobre la nueva plataforma.

Los COM accesados por clientes .Net son importados a .Net, en este proceso se crean los metadatos representando el interfaz COM, en la forma de clases de cubierta e interfaces, sin embargo, cierta características no tiene una adecuada conversión, por lo cual para mantener alta fidelidad entre la definición de la interfaz original COM y las clase de cubierta .Net, se debe asegurar que los componente de clase B usan interfaces compiladas de automatización OLE, de igual forma que no se deberán utilizar valores de retorno en métodos que puedan convertirse en ambiguos al realizar la conversión. De igual forma los componentes .Net que proveen servicios a clientes COM, es decir los clase C, tiene que ser expuestos al COM como objetos COM y sus interfaces, siendo esto un proceso de exportación, que se convierte en complicado debido a las limitaciones del COM, por lo cual en el resultado de esta, hay algunos tipos de componentes que pueden quedar estropeados y/o desordenados, tales como:

- Métodos sobrecargados.
- Métodos públicos que no son parte del interfaz.
- Métodos y campos públicos estáticos.
- Herencia múltiple de interfaces.

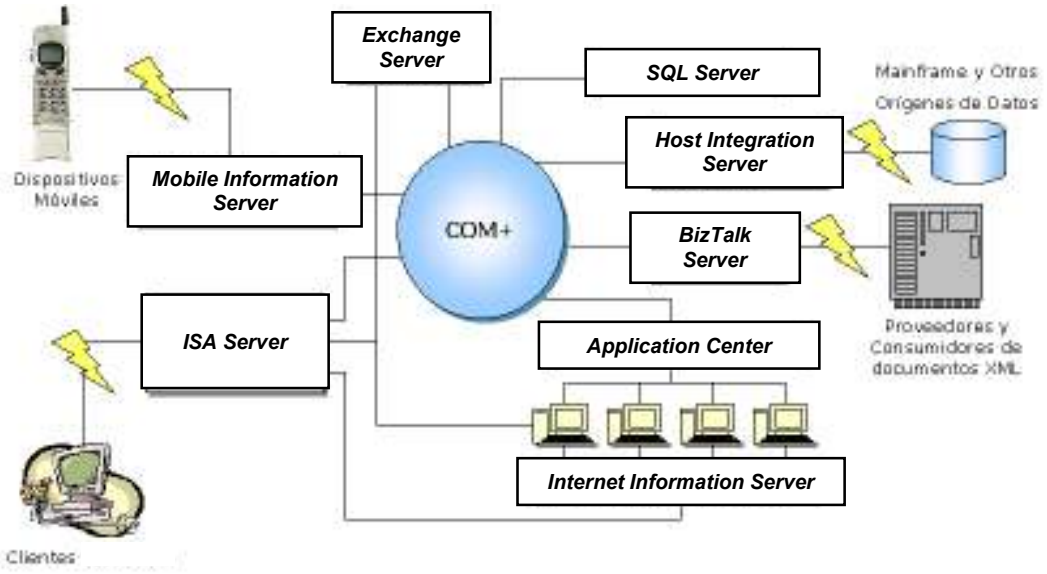
Un componente .Net que usa COM+ es llamado un **componente servido**, y un componente .Net que atiende a un componente COM+ es conocido como **empresa de servicios.** COM+ permite a los desarrolladores poder enfocarse sobre la lógica de negocios en lugar de en los componentes conectados, pudiendo disfrutar de una mejor productividad para hacer una aplicación robusta. En adición a esto el usar COM+ implica cierta arquitectura y estándares, y los componentes tienen que cumplir con estos requerimientos como los componentes tradicionales del COM., resultando de esto que la probabilidad de un buen ajuste entre los componentes existentes y los componentes servidos .Net es alta.

Otra parte importante de la interoperabilidad entre el COM+ y .Net, es el soporte de los servicios *Web*, siendo esto una importante característica de la plataforma .Net, ya que los servicios Web permiten una capa intermedia de componentes en un sitio *Web* para invocar métodos sobre otra capa intermedia de componentes en otro sitio *Web*, como si los dos componentes estuvieran sobre el mismo sitio. Para preservar la inversión existente en componentes COM y experiencia de desarrollo mientras se provea una ruta de migración al mundo de .Net, COM+ 1.5 puede exponer un componente COM+ que cumple con las guías de diseño de servicios *Web*, todo lo que se necesita para esto es especificar el directorio virtual de los servicios *Web* y COM+ proveerá lo necesario para adaptarlo como un componente de servicio.

Cada componente en una aplicación COM+ es expuesto como un servicio *Web* separado identificado por el **Prog-ID** del componente. COM+ instala los servicios *Web* con el Servidor de Información de Internet IIS, y genera la propia configuración del servicio Web, así como sus archivos de información, el único requerimiento es que IIS y .Net estén instalados sobre la máquina para habilitar el modo de activación de servicios *Web* para una aplicación.

Uno de los ejemplos más claros de la interacción de sistemas, es el planteamiento de Microsoft de una configuración de Servidor Empresarial .Net, en el cual se puede visualizar fácilmente la interoperabilidad entre diversos sistemas, así como la utilización del COM+ como eje de interacción entre estos. Tal como se ve en el gráfico la confección de la tecnología del Microsoft .Net, basado en XML, en el servidor empresarial de los productos Microsoft realza la escalabilidad, interoperabilidad, disponibilidad y manejabilidad que se requiere en un entorno complejo, en el cual se integran productos tales como *Windows Server, SQL Server, Aplication Center, BizTalk Server, Exchange Server, Comerse Server* y más.

Figura 5. Configuración de servidor empresarial .Net



1.4 El protocolo simple de acceso a objetos SOAP

SOAP, son la iniciales en inglés de *Simple Object Access Protocol*, y es un método para tener acceso a objetos remotos enviando mensajes XML, proporcionando independencia de plataforma y lenguaje, trabajando sobre varios protocolos de comunicaciones, siendo el más común HTTP, es un protocolo de comunicación ligero, y que es simple y extensible, por lo cual es utilizado para comunicación entre aplicaciones.

SOAP es la pieza clave de la tecnología .NET, pero al ser un protocolo abierto, permite usar todo su potencial desde cualquier lenguaje y plataforma, no es propiedad de Microsoft, sino es propuesto por la W3C (*World Wide Web Consortium*), pero aun no esta estandarizado. Cuando SOAP inició a principios de 1998, no existía un lenguaje de esquema o sistema de tipos para XML. El sistema inicial de SOAP tenía un puñado de tipos primitivos, compuestos que eran accesados por nombre y compuestos que eran accesados por posición. Una vez que estos tipos representativos estuvieron en su lugar, se modelaron tipos de comportamiento definiendo, es decir operaciones y métodos en términos de pares de estructuras, y cuando menos del lado de *DevelopMentor* y Microsoft, se agregaron estas operaciones dentro de las interfaces. Se puede decir que SOAP fué el primer intento de agregar a XML tipos de comportamiento. Las

propuestas existentes tanto asumían un sistema de tipos COM por debajo, razón por la que se estudio los protocolos de formato de serialización existentes como ASN.1 BER, NDR, XDR, CDR y TRMP, así como protocolos RPC (GIOP/IIOP, DCE/DCOM, RMI, ONC) y se trato de darle el sabor que agradaría al 80% elegantemente, pero que también fuera suficientemente flexible para adaptarse al 20% restante, sin embargo debido a políticas de Microsoft no se lanzó SOAP en dicho año.

Los colaboradores iniciales de SOAP dentro de Microsoft trabajaron en el equipo COM/MTS. Al mismo tiempo, el grupo SML dentro de Microsoft estaba laborando en XML-Data, lo que llegó a ser una de las muchas derivaciones del XML que se conocen actualmente. Como sucede frecuentemente en las grandes compañías, los dos grupos dentro de Microsoft no se podían ver, por lo que el apoyo público para SOAP se quedó atrapado dentro de Microsoft durante algún tiempo. No deseando que el lento proceso para lograr que Microsoft actuara en SOAP más allá de un anuncio en la prensa, Dave Winer (Parte del equipo de desarrollo de SOAP) por iniciativa propia lanzó la especificación XML-RPC, basada en el reestablecimiento del sistema de tipos de SOAP.

La segunda fase de desarrollo del SOAP, se puede decir que estuvo entre 1999 y el 2000, que fué el momento en que finalmente arrancó la especificación SOAP, utilizando el nombre **SOAP**, el lenguaje XML *Schema* W3C, aun no estaba terminado, pero ciertamente había progresado hasta el punto donde fué obvio para la mayoría de los autores de SOAP que era necesario reforzar e integrar lo posible, la labor del Grupo de Trabajo de *Schema*. Idealmente SOAP hubiera tomado el sistema ***Verbatim*** de tipos representativos del XML *Schemas* y simplemente hubiese agregado la idea de tipos de comportamiento y de métodos/operaciones, pero los XML *Schemas* carecían del apoyo para tipos sintéticos, tales como referencias y arreglos escritos. Mientras no se pueda definir en el esquema del lenguaje las cosas que parecen como referencias y arreglos, estas construcciones no son realmente originarias de los XML *Schemas*, por esa razón SOAP necesitaba aumentar el sistema de tipos con tipos ***soap: reference*** y

soap: Array. Es interesante observar que el Grupo de Trabajo de Schema trató de avocarse al tema de la referencia de tipo, pero desafortunadamente, no pudieron llegar a una solución que pudiera apoyar las referencias de tipo conforme aparecían en la mayoría de los sistemas de tipo programático.

Posiblemente el mayor problema técnico que enfrentaba SOAP ente 1999 y el 2000, era la falta de metadatos. *DevelopMentor*, trató de introducir un formato sencillo de metadatos CDL que era isomórfico con el sistema de tipos de XML *Schema*. *Dave Winer* y *Eric Raymond* (Parte del equipo de creación del SOAP), visualizaban que como metadatos lo que se necesitaba eran descripciones legibles para el humano, lo cual concordaba con la finalidad del CDL, sin embargo *Gopal Kakivaa* de Microsoft, convenció al equipo de desarrollo de que lo que se necesitaba, podía lograrse anotando XML *Schemas* con tipos adicionales específicos de SOAP que fueron autorizados (y de hecho anticipados) por la especificación de Schema. En este punto, *DevelopMentor* se unió a *Schemas* WG y la mayor parte del esfuerzo de desarrollo se movilizó internamente hacia el soporte del XML *Schema*.

La especificación XML *Schema* es estable, según *Don Box* (Parte del Equipo de Desarrollo de SOAP), “Este es el avance más importante para la gente que se preocupa por los protocolos XML y la mensajería en general, y por SOAP en especial. El hecho de que no se pueden hacer cambios mayores antes de avanzar a una completa recomendación de W3C, significa que en general la industria sabe con quién está luchando cuando resulta que aplica tipos a XML. Lo he dicho antes y aún lo sostengo, que sin XML *Schemas*, XML sería un estándar petrificado y sería bastante menor su utilidad para el software, los componentes o el servicio de integración.” Actualmente W3C tiene un grupo de trabajo para el protocolo XML, por lo cual SOAP está ahora en el lugar apropiado, puesto que ha sido clasificado dentro del trabajo del protocolo XML, lo que ha llevado a que actualmente se este

bastante cerca de tener un formato estandarizado de metadatos, de igual forma las ideas de SOAP han sido aceptadas casi por todo mundo en este punto.

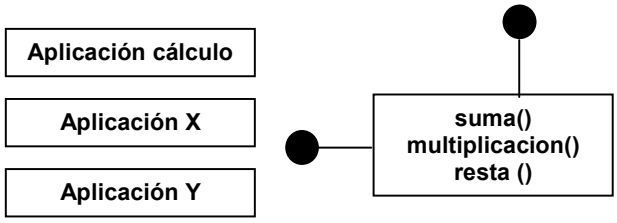
Algunas de las cosas que se deben de tener claras acerca del SOAP son:

- Los mensajes son enviados de forma conocida, requerimiento / respuesta.
- SOAP define una estructura XML para llamar un método y pasarle sus parámetros.
- SOAP define una estructura XML para que el método retorne los valores.
- SOAP define una estructura XML para devolver códigos de error, si el proveedor del servicio no puede ejecutar el método requerido.
- SOAP no define como el solicitante o el receptor envían y reciben mensajes.
- La implementación del envío y la recepción de mensajes esta a cargo del desarrollador del software.
- SOAP no define como una vez recibido el mensaje, se crean instancia de objetos y ejecución.

1.4.1 Funcionamiento del SOAP

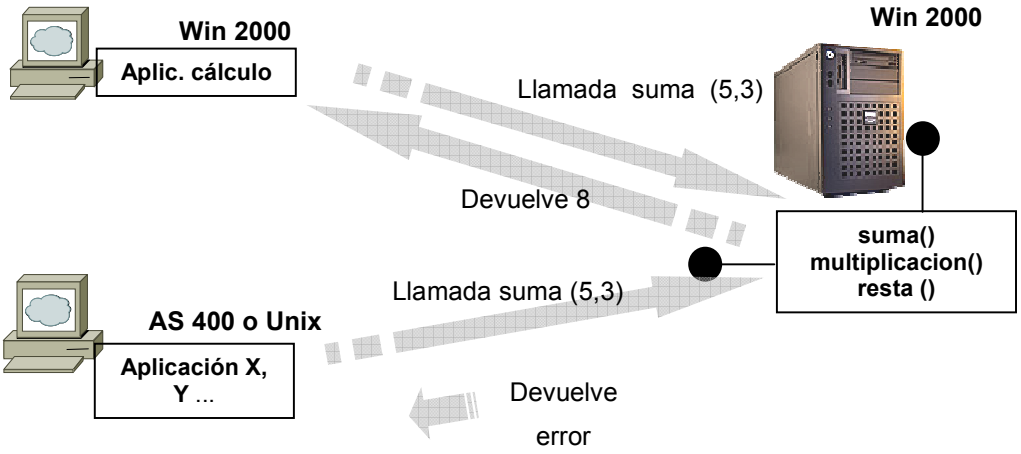
Cuando se está desarrollando una aplicación, lo que se hace normalmente es que se crean procedimientos que proveen la funcionalidad que se necesita, y cuando se tienen procedimientos que están relacionados, estos se agrupan dentro de un contenedor, es decir un objeto. Por ejemplo aquí tenemos procedimientos relacionados con funciones matemáticas. La idea general de esto es que cuando se desarrollan los objetos, lo que se pretende es que varios programadores lo utilicen y también en diferentes aplicaciones sobre cualquier plataforma.

Figura 6. Relación aplicaciones y objetos de servicios



Las condiciones ideales mencionadas casi nunca se cumplen, y si desarrolla un objeto sobre una plataforma X, el objeto queda atado a ella y no podrá ser utilizado por aplicaciones en otras plataformas.

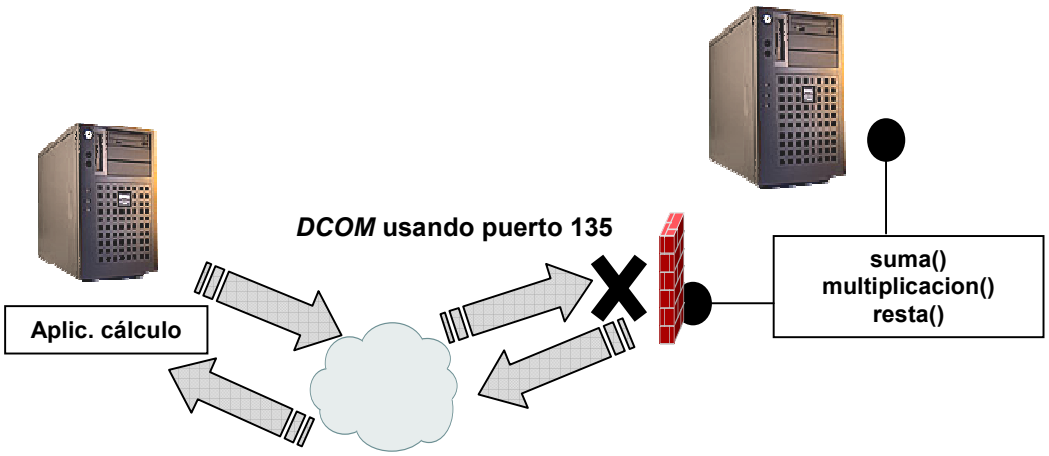
Figura 7. Uso de objetos por diversas plataformas y lenguajes



El problema típico de la ejecución de procedimientos remotos, como CORBA y DCOM, está en la comunicación, principalmente por la existencia de *Firewalls*, ya que esto complica bastante el intercambio de información, en los casos de DCOM que utiliza puertos específicos para ello; sin embargo como SOAP trabaja sobre HTTP, el problema se minimiza, pues este protocolo

se maneja bien con estos dispositivos, asimismo si se quiere que la comunicación sea segura se puede utilizar el protocolo HTTPS en lugar del simple HTTP.

Figura 8. Comunicación y ejecución de métodos remotos



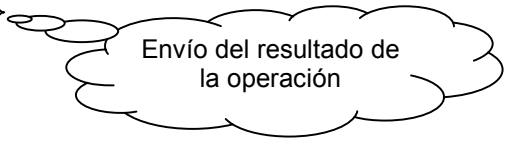
Para ejemplificar con código lo expuesto acerca de las aplicaciones, utilizaremos la aplicación **cálculo**, la cual esta llamando un método del objeto, **suma()**, la aplicación **cálculo** armaría el mensaje SOAP con la siguiente estructura:

```
<?xml version="1.0"?>
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
    <SOAP:Body>
      <suma>
        <primernumero>    5  </Primernumero>
        <segundonumero> 3  </Segundonumero>
      </suma>
    </SOAP:Body>
</SOAP:Envelope>
```

Llamada a método y paso de parámetros

El mensaje anterior llega al servidor, el cual realiza el procesamiento necesario con este. Terminado el procesamiento, el servidor construye la respuesta que es otro mensaje SOAP, que tendría la siguiente estructura:

```
<?xml version="1.0"?>
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
    <SOAP:Body>
        <SumaRespuesta>
                <Valor>  8  </ Valor >
        </SumaRespuesta>
    </SOAP:Body>
</SOAP:Envelope>
```



Esto sería un proceso de comunicación básico con mensajes SOAP. El medio que se está utilizando es HTTP, este es el preferido porque emplea el puerto 80 del *firewall*. SOAP también funciona con todos los protocolos existentes sobre Internet.

La estructura de un mensaje SOAP consta de tres elementos principales:

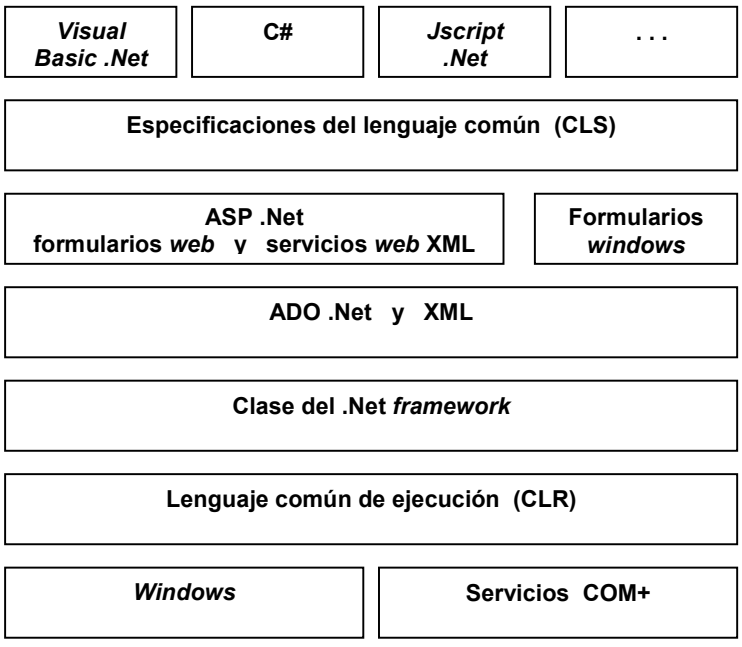
1. **Cubierta (e*nvelope*)**, que es el elemento en el que se define el contenido del mensaje.
2. **Encabezado (h*eader*)**, que es opcional y contiene la información sobre el encabezado, por medio de este se pueden agregar características al mensaje SOAP.
3. **Cuerpo (*body*),** es el que contiene llamados e información de respuesta, utilizando dentro de este el elemento llamado ***Fault*,** para informar de los errores.

2. EL MICROSOFT .NET *FRAMEWORK*

2.1 Generalidades del .Net *Framework* y a su arquitectura

El .Net *Framework* está diseñado para permitir programación basada en *Web* y para programación tradicional, para construir aplicaciones más eficientes y habilitar de esta forma un trabajo más flexible. Una de las características más importantes del *Framework* es que habilita el código escrito en múltiples lenguajes para trabajar en forma conjunta.

Figura 9. Arquitectura del .Net *Framework*



La arquitectura del *Framework*, se basa principalmente en el manejo de estructuras que puedan ser compiladas hacia un lenguaje intermedio que puede ser interpretado por un lenguaje común de ejecución (del cual se hablará más a detalle más adelante en este capítulo), siendo estos los archivos conocidos como ejecutables portátiles o PEs, con la finalidad de que este tipo de archivo sea soportado por otros sistemas operativos diferentes a Microsoft Windows en el futuro. Para entender como se integra este tipo de programas, se deben saber las partes de un módulo manejado:

- **Encabezado PE (PE *header*)**, este es un encabezado similar al formato común de archivos de objetos COFF. Indica el tipo de archivo GUI, CUI o DLL, y este también tiene un sello de tiempo indicando cuando fue construido el archivo. Para módulos que contienen solo código IL (Lenguaje Intermedio), el volumen de la información en el encabezado PE es ignorado. Para módulos que contiene código nativo de CPU, el encabezado contiene información acerca de este código.

- **Encabezado CLR (CLR *header*)**, contiene la información (interpretada por el CLR y sus utilidades) que hace a este código un módulo manejable. El encabezado incluye la versión del CLR requerido, algunas banderas, los metadatos de definición de métodos, así como la localización y tamaño de los metadatos de los módulos, los recursos, nombres de almacenamiento y otro material menos importante.

- **Metadatos,** cada módulo manejado contiene metadatos de las tablas, siendo de dos tipos principales, el primero, son tablas que describen los tipos y la definición de miembros en el código fuente, y el segundo, son tablas que describen los tipos y miembros referenciados por el código fuente.

- **Código de lenguaje intermedio**, es el código que el compilador produce como un código fuente compilado. El CLR posteriormente compilara el IL en instrucciones de código nativo de CPU.

Los lenguajes existentes en la actualidad, en su mayoría generan código con arquitectura para un específico CPU, tal como los x86, IA64, Alpha, etc., mientras que los compiladores de CLR producen un código intermedio en lugar de código nativo, en adición a esto, cada compilador del CLR requiere que se genere completamente los metadatos en cada módulo manejado, en pocas palabras los metadatos son simplemente un grupo de tablas de datos que describen que esta definido en los módulos, tal como tipos y sus miembros, asimismo también tiene tablas de indicaciones de las referencias a los módulos manejados, siendo así los metadatos un grupo de viejas tecnologías como bibliotecas de tipos y archivos de lenguajes de definición de interfaces IDL. Sin embargo la parte importante de esto es que los metadatos CLR son más completos, y a diferencia de las bibliotecas de tipos y los IDL, los metadatos están siempre inmersos como código en el mismo archivo EXE o DLL (por ejemplo), haciendo esto imposible la separación de ambos, porque el compilador al procesar los metadatos y el código al mismo tiempo, los une dentro del resultando del módulo manejado, y estos junto con el código IL están descritos y no están nunca fuera de sincronización uno con el otro.

Usualmente los desarrolladores podrán programar en lenguajes de alto nivel tales como C# o *Visual Basic*, pero el compilador de estos lenguajes produce IL, sin embargo como cualquier otro lenguaje de máquina, IL puede ser escrito en lenguaje ensamblador, y Microsoft provee este ensamblador IL *Assembler* (del cual hablaremos en el siguiente capítulo) cuyo archivo ejecutable es el ILAsm.exe, y de la misma forma también provee un desensamblador IL *Disassembler* cuyo archivo ejecutable es el ILDasm.exe.

La parte más importante dentro de la arquitectura del .Net *Framework* es el Lenguaje Intermedio, que ha sido considerado inseguro en cuanto a protección de la propiedad intelectual de los algoritmos, esto debido a herramientas como el desensamblador, con las que fácilmente se puede realizar ingeniería de reversa para obtener en forma exacta el código de una aplicación, lo cual es muy cierto, sin embargo cuando se implementan sistemas basados en servicios XML o *Web Forms*, el código de los módulos manejados reside únicamente en el servidor, porque aunque personas fuera de la compañía puedan acceder al módulo, estas no podrán utilizar un herramienta que les permita ver el IL.

Es importante mencionar que en Octubre del año 2000, Microsoft propuso un gran grupo del .Net *Framework* a la ECMA (*European Computer Manufacture's Association*), con el propósito de la estandarización, y la ECMA acepto esta proposición y creo la comisión técnica (TC39) para observar el proceso de estandarización. El comité técnico está a cargo con las siguientes obligaciones:

- **Grupo 1**, responsable de desarrollar un lenguaje dinámico estándar de scripts (*ECMAScript*). La implementación de Microsoft son los JScripts.
- **Grupo 2**, es encargado de desarrollar una estandarización de versiones de el lenguaje de programación C#.
- **Grupo 3**, es responsable de desarrollar un lenguaje de infraestructura común CLI. Específicamente, el CLI definirá un formato de archivo, un sistema común de tipos, un sistema de metadatos extensible, un lenguaje intermedio, y acceso a una plataforma fundamental. Conjuntamente con esto, el CLI definirá una factible base de biblioteca de clases, diseñada para ser usada por múltiples lenguajes de programación.

Una vez la estandarización este completa, este estándar será distribuido por ISO/IED JTC 1 (Tecnología de Información). En este tiempo, el comité técnico también investigará futuras direcciones para el CLI, C#, y los ECMAScripts, tal como para cualquier propósito complementario o tecnología adicional. El .Net *Framework* ofrece una gran cantidad de ventajas sobre otras plataformas de desarrollo, sin embargo, muchas compañías pequeñas pueden tener que rediseñar y reimplementar todo el código ya existente. Microsoft tiene estructurado el CLR de tal forma que este ofrece mecanismos que permiten una aplicación consistente para código manejado y no manejado. Específicamente CLR soporta tres escenarios de interoperabilidad de código:

1. **Código manejado que puede llamar a funciones de código no manejado en un DLL**, en este escenario, el código manejado puede hacer llamadas a funciones contenidas en DLLs, utilizando el mecanismo ***P/Invoke***. Muchos de los tipos definidos en el FCL internamente llamadas funciones exportadas, se pueden hacer desde Kernel32.dll, User32.dll, etc. Muchos lenguajes de programación podrán exponer un mecanismo que haga esto fácil para el código manejado, para llamar a las funciones de código no manejado.

2. **Código manejado que puede usar un componente COM existente**, este escenario debe ser considerado debido a que muchas compañías tienen ya implementados una gran cantidad de código no manejado. Usando la librería de tipos desde este componente, un ensamblado manejado puede ser creado para describir el componente COM. El código manejado puede acceder a los tipos en el ensamblado manejado solo como otro tipo manejado.

3. **Código no manejado que puede usar tipos manejado**, en este caso un lote existente de código no manejado requiere que se le provea con un componente COM desde el código para trabajar correctamente. Esto es muchas veces fácil de implementar, usando código manejado así que se pueda ignorar todo el código que se tiene y hace la referencia, esto con contadores de referencia e interfaces. Por ejemplo se podría crear un control ActivX o una extensión *shell* en C# o *Visual Basic*.

2.2 Dentro del *Framework*

2.2.1 Los ensamblados

En la actualidad el CLR no trabaja con módulos de código, sino trabaja con **ensamblados**, siendo esto un concepto un poco abstracto difícil de entender inicialmente, pero se puede decir que este es una agrupación lógica de uno o más módulos manejados o archivos de recursos, y también que es la más pequeña unidad de código de reutilización, seguridad y versionable, dependiendo sobre la elección que se haga del compilador o herramienta, se puede producir un archivo simple o un ensamblado multiarchivos. Los ensamblados, son una parte fundamental de la programación con el .Net *Framework*, realizando funciones tales como:

- Contener código que ejecuta el CLR. El lenguaje intermedio de Microsoft MSIL utiliza los archivos ejecutables portátiles o PEs, pero no serán ejecutados si no tiene asociado un manifiesto de ensamblado, haciéndose la aclaración que cada ensamblado puede tener un único punto de partida, tal como un *DllMain, WinMain* o *Main*.
- El ensamblado forma una frontera de seguridad, ya que este es la unidad en la cual los permisos son solicitados y otorgados.

- Forma una frontera de tipos, ya que cada identificador de tipo incluye el nombre del ensamblado en el cual reside.
- Forma una frontera de extensiones referenciadas, dado que el manifiesto de ensamblado contiene los metadatos que son usados para resolución de tipos y satisfacción de requerimientos de recursos, también especifica los tipo y los recursos que son expuestos fuera del ensamblado, de igual forma que enumera otros ensamblados sobre los cuales está dependiendo.
- Forma una frontera de versiones, ya que el ensamblador es la más pequeña unidad versionable en el CLR; todos los tipos y recursos en el mismo ensamblado son versionados con una sola unidad, y en el manifiesto se describe dicha versión y la dependencia de versiones específicas para cualquier ensamblado dependiente.
- Forma una unidad de distribución. Cuando una aplicación inicia, solo los ensambladores de la aplicación son inicialmente llamados, otros ensamblados, tales como localización de recursos o ensamblados que contienen clases de empleo, pueden volver a ser demandadas.
- Es la unidad que es soportado en una ejecución lado a lado.

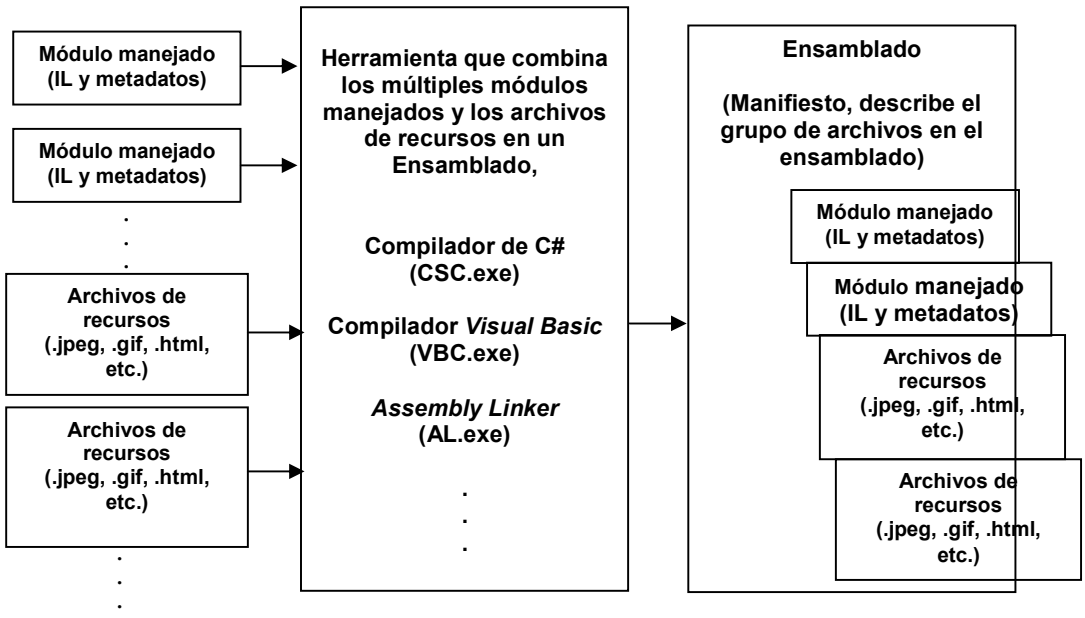
Los ensamblados pueden ser estáticos o dinámicos, los estáticos pueden incluir tipos de .Net *Framework* (interfaces y clases), tal como recursos para el ensamblado (*bitmaps*, archivos *jpeg*, etc.), estos son almacenados sobre disco en un archivo PE. Se puede usar el .Net *Framework*, para crear ensamblados dinámicos, que corren desde memoria y no son grabados a disco antes de ejecutarse. Uno de los modos para crear ensamblados, es utilizar herramientas de desarrollo como *Visual Studio* .Net, pero también pueden ser usadas herramientas proveídas por el .Net *Framework* SDK, con el que se pueden crear ensamblados con módulos creados en otros entornos de desarrollo.

En general, un ensamblado estático puede consistir de cuatro elementos, el **manifiesto**, que contiene los metadatos, **metadatos de tipos**, **código MSIL**, que

es donde se implementan los tipos, y finalmente un **grupo de recursos**. Solo el manifiesto de un ensamblado es requerido, pero cualquier tipo o recurso que se necesite para dar al ensamblado un sentido de funcionalidad.

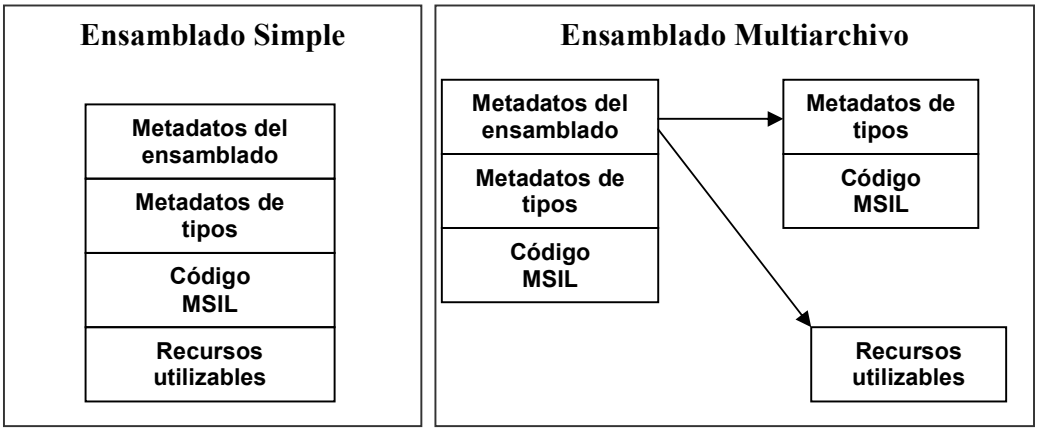
En la figura siguiente se puede observar que es un ensamblado, en esta se ven los módulos manejados y los archivos de recursos que son procesados por una herramienta, la cual produce un archivo PE, que representa la lógica del grupo de archivos. Como se puede ver el archivo PE contiene un bloque de datos llamada manifiesto, siendo este simplemente otro grupo de tablas de metadatos, en ellas se describen los archivos con los que se construyó el ensamblado, los tipos exportados públicamente que se implementaron por los archivos en el ensamblado, y finalmente los recursos o archivos de datos que son asociados con el ensamblado.

Figura 10. Combinación de módulos manejados dentro de un ensamblado



Como ya se mencionó, los ensamblados se pueden realizar como un archivo simple o alternativamente como multiarchivo, en este caso los archivos pueden ser módulos de código compilado (llamados .*NetModule*), archivos de recursos, u otros archivos requeridos por la aplicación. El crear un ensamblado de múltiples archivos es una buena opción cuando se quiere combinar módulos escritos en diferentes lenguajes y para optimizar la descarga de una aplicación que rara vez son utilizadas fuera de línea y usan tipos en un módulo que son descargados solamente cuando son necesitados.

Figura 11. Ensamblado simple y multiarchivo



Como ya se mencionó antes, una parte importante del ensamblado es el manifiesto del mismo, el cual realiza funciones tales como la enumeración de archivos que sirvieron para crear el ensamblado, asimismo es el encargado de regir a los tipos del ensamblado y el mapeo de los recursos para los archivos que contienen sus declaraciones e implementaciones. Otra actividad del manifiesto es enumerar otros ensamblados sobre los que hay dependencia, así como proveer un nivel indirecto entre los clientes del ensamblado y los detalles de implementación de este.

El contenido del manifiesto es el siguiente:

- **Nombre del ensamblado**, que es un texto en el que especifica dicho nombre.
- **Número de versión**, contiene el número de una mayor y una menor versión, así como un número de revisión y construcción. El CLR utiliza este número para forzar las políticas de las versiones.
- **Cultura**, es un campo de información sobre el lenguaje que soporta el ensamblado, siendo esta información solo al diseñar un ensamblado como un ensamblado satélite, conteniendo cultura de información de un lenguaje específico. Un ensamblado con cultura de información es automáticamente asumido para ser un ensamblador satélite.
- **Información de nombre fortalecido**, si el ensamblado esta dando un nombre fortalecido, el manifiesto contiene una llave pública desde la cual lo publica.
- **Lista de todos los archivos en el ensamblado**, esto es una lista de cada uno de los archivos contenidos en el ensamblado. Los archivos que sirvieron para crear el ensamblado deben estár en el mismo directorio que el archivo que contiene el manifiesto.
- **Información de refencia de tipos**, esta información es usada por el CLR para mapear un tipo referenciado al archivo que contiene su declaración y su implementación. Este es usado por tipos que son exportados desde otro ensamblado.
- **Información sobre ensamblados referenciados**, es una lista de otros ensamblados que son estáticamente referenciados por este ensamblado. Cada referencia incluye el nombre del ensamblado referenciado, los metadatos del ensamblado tales como versión, cultura, sistema operativo, etc., y una llave pública en el caso de que el ensamblado tenga un nombre fortalecido.

Respecto al contenido del manifiesto es importante mencionar que los primeros cuatro atributos del manifiesto, son los que se utilizan para la formación del identificador del ensamblado, así como especificar que se pueden realizar cambios en la información de este manifiesto usando atributos en código fuente.

2.2.2 Manejo de tipos comunes

Una definición de tipos constituye la construcción de un nuevo tipo desde uno ya existente. La construcción de los valores para los tipos, punteros, arreglos y delegados son definidos cuando son usados y son referenciados como tipos implícitos, asimismo pueden ser anidados, es decir que puede ser miembro de otro tipo. Una definición de tipos incluye:

- **Cualquier atributo definido sobre los tipos**, son etiquetas declarativas que proveen información adicional en tiempo de ejecución, estos pueden ser aplicados a casi cualquier lenguaje elemental (tipos, propiedades, métodos, etc.).

- **La visibilidad del tipo**, todos los tipos tienen una declaración de esto, los tipos anidados tienen miembros que pueden ser accesibles, según su visibilidad. En el CLR existen dos tipos de visibilidades, pública que permite que el tipo sea visible por todos los ensamblados, y la de ensamblado, en la que el tipo es visible solamente dentro del ensamblado.

- **El nombre del tipo**, generalmente estos nombres no son únicos, son agrupados entre un enmarcado, entre estos un nombre puede ser referenciado por múltiples entidades tanto como ellos sean diferentes o tiene diferentes firmas. El sistema de tipos comunes incluye entidades tales como tipos, métodos, campos, tipos anidados, propiedades y eventos. Los tipos proveen un enmarcado para sus nombres. Todas las comparaciones e identificaciones en este esquema de tipos son realizadas byte por byte, y son de tipo sensitivo (*Case Sensitive*). Los nombres de tipos necesitan solamente ser único dentro del ensamblado. Para identificar completamente un tipo, los nombres de tipo pueden ser calificados por el enmarcado que lo incluye, así como por el ensamblado que contiene su implementación.

- **El tipo base para generar el nuevo tipo**, un tipo puede heredar valores desde otro, algunos términos comunes para los tipos que reciben la herencia son los tipos hijos, subtipos, y tipos derivados y los términos comunes para el tipo que hereda son tipo padre, supertipo, y tipo base. El tipo base generalmente se refiere a el tipo directo desde el que se esta teniendo herencia, mientras que el tipo padre y el supertipo puede referenciar tanto al tipo del cual se hereda directamente, o referencia a cualquiera de sus padres.

Tabla I. Definición de mienbros de los tipos

Miembro	Descripción
Eventos	Define un incidente que puedes ser contestado, y define métodos para cubrirlo, descubrirse, y disparado del evento. Los eventos son frecuentemente usados para informar a otro tipo de un cambio de estado.
Campos	Describe y contiene el valor de un tipo. Los campos pueden ser de cualquier tipo soportado por el CLR.
Tipos anidados	Define un tipo entre la esfera del tipo rodeado.
Métodos	**Describe operaciones disponibles sobre el tipo. La firma del método especifica los tipos permisibles de todos estos argumentos y de esto retorna valores.** Un constructor es una bondad especial del método que inicializa la nueva instancia de un tipo.
Propiedades	Nombre un valor lógico o el estado del tipo y métodos definidos por dar o colocar los valores de las propiedades. Las propiedades son frecuentemente usadas para mantener las interfaces públicas de un tipo independiente desde la representación del tipo actual.

El sistema de tipos comunes, permite algunas características especiales para algunos de los miembros de cada tipo, sin embargo no todos los lenguajes las necesitan o las soportan.

Tabla II. Carácterísticas de miembros de los tipos

Característica	Aplica	Descripción
Abstracto	Métodos, propiedades y eventos	Es aplicable para miembros que se desea no puedan ser utilizados en forma directa para la clase, es decir solamente podrán ser utilizados por una clase que los herede, pero no al hacer conexiones a la clase original directamente.
Privado, familiar, ensamblado, familia o ensamblado, familia y ensamblado o público.	Todos	Define la accesibilidad de los miembros. Privado, es cuando son accesibles solo desde dentro del mismo tipo como el miembro o en tipos anidados. Familiar, es cuando son accesibles desde dentro del mismo tipo con el miembro y desde subtipos que son heredados desde aquí. Ensamblado, es accesible solamente desde la misma unidad de compilación, sujeto a grupos de reglas para un lenguaje fuente específico. Público, es accesado desde cualquier otro tipo.
Final	Métodos, eventos y propiedades.	Los métodos virtuales no pueden ser invalidados en un tipo derivado.
Iniciación	Campos	Un valor puede solo ser asignado a el campo entre un bloque de inicialización.
Instancia	Campos, métodos, propiedades y eventos	Los miembros que no son ni estáticos ni virtuales son llamados de esta forma.

Literal	Campos	El valor asignado al campo es un valor corregido, sabiéndolo en tiempo de compilación. Los campos literales son en ocasiones referenciados como constantes.

Continuación

Característica	Aplica	Descripción
Newslot o Override	Todos	Define como los miembros interactúan con miembros heredados de la misma firma. Newslot, oculta los miembros heredados con la misma firma. Override o Invalidar, reemplaza la definición de un método.
Estático	Campos, métodos, propiedades y eventos	El miembro pertenece a el tipo sobre el que es definido, no a una instancia en particular o tipo; el miembro es compartido entre todas la instancias del tipo.
Virtual	Métodos, propiedades y eventos	El método puede ser implementado por un subtipo y puede ser invocado estática o dinámicamente.

Cada miembro del tipo tiene una firma única. La firma de los métodos consisten en el nombre del mismo, una lista de parámetros (el orden y tipo de los argumentos del método) y el tipo del valor de retorno del método. Más que un método con el mismo nombre puede definirse dentro de un tipo mientras su firma sea diferente. Cuando dos o más métodos con el mismo nombre son definidos, se dice que el método está **sobrecargado**.

Un tipo hijo hereda todos los miembros de su tipo padre, esto es que los miembros son definidos sobre el padre y son disponible para el tipo hijo. Un tipo hijo, puede ocultar un miembro heredado para definir un nuevo miembro con la misma firma, esto se podría hacer para un método heredado previamente a definir un nuevo comportamiento para un miembro público o privado como el ser marcado como final. De igual forma un tipo hijo también puede invalidar un método virtual heredado. La invalidación de métodos provee una nueva definición de los métodos que pueden ser invocados basados sobre el tipo del valor en tiempo de ejecución, tanto que el tipo de la variable se sabe en tiempo de compilación. Un método puede invalidar un método virtual solamente si el método virtual está marcado como final y el nuevo método esta al menos como accesible de la misma forma que el virtual.

La parte esencial del sistema de tipos comunes del .Net *Framework* es el manejo de clases, lo cual es familiar si se conoce la programación orientada o objetos, como se sabe las clases definen las operaciones en un objeto que puede ejecutar métodos, eventos, etc., y define un valor del estado de el objeto. Aunque una clase generalmente incluye ambas definiciones e implementaciones, esto puede tener uno o más miembros que no tienen implementación. Una instancia de una clase .Net *Framework* es un objeto, y se puede acceder a una función de objetos para llamar a métodos y a sus propiedades, eventos y campos.

Cada lenguaje electo tiene sintaxis propia para crear instancias de clases, pero cumplen con algunas características que el CLR permite en las clases.

Tabla III. Características Especiales de los Tipos

Característica	Descripción
Sellado	Especifica que otro tipo no puede ser derivaros desde este tipo.
Instrumento	Indica que la clase cumple el contrato especificado por uno o más interfaces.
Abstracto	Especifica que no se puede crear una instancia de la clase. Para usar esto, se podría derivar otras clases desde esta.
Herencia	Indica que instancias de la clase, pueden ser usadas dondequiera a la clase base especificada. Una clase derivada que se hereda desde una clase base puede usar los instrumentos de cualquier método virtual provisto por la clase base, de igual forma que la clase derivada puede invalidarlos con sus propios instrumentos.
Exportados o no exportados	Indica si una clase es visible fuera del ensamblado en el cual está definido. Aplica solo para niveles de clase alta.

Las clases anidadas, también tienen características de miembros. Los miembros de las clases que no tienen instrumentación son los miembros abstractos. Una clase que tiene uno o más miembros abstractos son por si mismos abstractos. Se pueden utilizar clases abstractas cuando se necesita encapsular un grupo base de funciones de esas clases derivadas que pueden heredar o invalidar las funciones cuando sea apropiado. Las clases que no son abstractas son referenciadas como clases concretas.

2.2.3 Metadatos de componentes

En el pasado, un componente de software, se escribía en un lenguaje que podía ser fácilmente usando un componente escrito en otro lenguaje, COM proveía un paso siguiente en solventar este problema, ahora el .Net *Framework* hace interoperabilidad de componentes fácil, permitiendo compilados para emitir declaraciones adicionales de información dentro de todos los módulos y ensamblados. Esta información, es llamada metadatos, y ayuda a los componentes por si mismos a interactuar.

Los metadatos es información binaria describiendo los programas que están almacenados en archivo PE para el CLR o en memoria. Cuando se hace la compilación de código, los metadatos son insertados dentro de una porción del archivo, mientras que el código es convertido al MSIL y se inserta en la otra porción del archivo. Cada tipo y miembro definido y referenciado en un módulo o ensamblado es descrito dentro de los metadatos. Cuando el código es ejecutado, el CLR carga los metadatos a memoria y referencia estos para descubrir la información acerca de las clases del código, así como los miembros, la herencia, etc. Los metadatos almacenan la siguiente información:

- **Descripción del ensamblado**
  - Identificación
  - Los tipos que serán exportados.
  - Otros ensamblados de los cuales depende este.
  - Permisos de seguridad que se necesitan para correr.

- **Descripción de tipos**
  - Nombre, visibilidad, clase base e instrumentos de interfaces.
  - Miembros (métodos, campos, propiedades, eventos y tipos anidados).
- **Atributos**
  - Elementos descriptivos adicionales que modifican tipos y miembros.

2.2.3.1 Beneficios de los metadatos

Los metadatos son la clave para un modelo de programación simple, eliminando la necesidad de archivos de lenguajes de definición de interfaces, IDLs, archivos encabezados o cualquier método externo de referencia de componentes, ya que pueden describir por si mismos el código desarrollado. La descripción de archivos para módulos y ensamblados en el CLR se realiza a través de los metadatos, estos contienen todo lo necesario para interactuar con otros módulos, proveyendo automáticamente la funcionalidad de un IDL en los COM, permitiendo el uso de un solo archivo para las definiciones y los instrumentos. Los módulos y los ensamblados, no requieren registro con el sistema operativo, facilitando la distribución de aplicaciones. Como resultado de esto, las descripciones usadas por el CLR permiten reflejar el código actual en el archivo compilado, lo cual incrementa la confiabilidad de las aplicaciones.

La interoperabilidad de lenguajes y fácil diseño basado en componentes es provista por la información de los metadatos, que es requerida en el código compilado para una clase heredada, desde un archivo PE escrito en diferentes lenguajes. Se puede crear una instancia y una clase escribiéndola en lenguajes manejados por el CLR, sin inquietarse acerca de administración explícita o usando el habito de la interoperabilidad de código.

2.2.4 Dominios de aplicación

Cuando el servidor CLR COM son cargados dentro de un proceso Windows, estos son inicializados, y parte de esta inicialización es para crear el manejo de la memoria de pila "*Heap*" a la cual reverenciaran los objetos obteniendo localización desde la colección de desperdicios. Adicionalmente a esto, el CLR crea una capa usada para cualquiera de los tipos manejados cuyos ensamblados son cargados dentro del proceso. Mientras se realiza la iniciación, también crea lo que es conocido como dominio de aplicación, el cual es un contenedor lógico para un grupo de ensamblados. El primer dominio al inicializar CLR es el *default*, el cual puede ser anfitrión de instrucciones del CLR para crear dominios adicionales, además el código en el ensamblado manejado puede también decirle al CLR que cree otros dominios. Las tres características de las que hacen uso los dominios de aplicación son:

1. **Los dominios son aislados uno de otro**, ya que un dominio puede ver los objetos creados por un diferente dominio. Esto fortalece una limpia separación porque el código en un dominio puede hacer una referencia directa hacia un objeto creado en un dominio diferente. Este aislamiento permite a los dominios tener una fácil descarga desde un proceso.

2. **Los dominios pueden ser descargados**, esto es debido a que el CLR no soporta la habilidad de descargar un ensamblado simple. Sin embargo le puede decir al CLR que descargue un dominio y todos sus ensamblados actualmente contenidos en él.

3. **Los dominios pueden ser configurados y asegurados individualmente**, ya que cuando se crea un dominio puede tener evidencia aplicada a este. La evidencia es una característica relacionada con la seguridad que determina los derechos máximos otorgados para un ensamblado corriendo en el dominio. Más común, es que un dominio tendrá una política de seguridad aplicada a el uso de los métodos en él.

Algunas de las configuraciones dentro de un dominio de aplicaciones son:

- **Nombre de aplicación (a*plicationname*)**, es una cadena de caracteres con el nombre de la aplicación, que es usado para identificar el Dominio de Aplicaciones.
- **Base de aplicación (*aplicationbase*)**, que es un directorio donde el CLR mirará para localizar los ensamblados.
- **Ruta privada bin (*privatebinpath*)**, que es un grupo de directorios donde el CLR mirará para localizar livianamente los nombres de los ensamblados.
- **Archivo de configuración (*configurationfile*)**, el nombre de la ruta de un archivo de configuración contiene las reglas que el CLR utilizará para localizar los ensamblados. El archivo también contiene configuraciones remotas, como configuraciones de aplicaciones *Web*, etc.
- **Optimización de carga (*loadoptimization*)**, es una bandera que le dice al CLR si se tratará de cargar el ensamblado como dominio neutral o como un dominio simple.

Algo importante, es que lo anterior podría ser degenerado para correr múltiples aplicaciones no manejadas en un proceso simple. La razón de esto es que diferentes aplicaciones tienen accesos a cada dato de otra y a su código, haciendo todo también más fácil para una aplicación el corromper la otra. Sin embargo, no es una negociación con el código manejado porque el código IL manejado es del mismo tipo y este es verificado, haciendo imposible para el código en un dominio que corrompa el de otro. Por supuesto, un administrador podrá turnar verificaciones a distancia y permitir que el código manejado haga llamadas a funciones de código no manejadas.

El código en un dominio de aplicaciones puede comunicarse con tipos y objetos contenidos en otro dominio, sin embargo el acceso a estos tipos y objetos es solamente a través de un mecanismo bien definido. Más tipos son manejados por valor cruzando las fronteras de los dominios. El dominio destino usa la referencia de este nuevo objeto, ya que en realidad no tiene acceso al original. Por objetos para ser remotamente por valor, el tipo de objeto tendrá que tener los atributos de serialización (*System.Serializable*) aplicados a él.

Cuando una referencia a objetos es pasada para un dominio destino, el CLR crea una instancia de tipo *Proxy* en este dominio, y una referencia a este objeto Proxy es en la que el código será utilizado por el dominio destino. El objeto original y sus campos quedan en el dominio original, ya que el objeto proxy es una cubierta que sabe como son las llamadas a métodos sobre el objeto original en el dominio original. Otra vez, el destino no tiene acceso directo al objeto en el dominio original. Obviamente, el accesar objetos cruzando los límites de los dominios tiene costo en la respuesta de ejecución, por lo que cuando sea posible es mejor intentar tener lo más nulo posible el uso de objetos cruzando estos límites.

Tabla IV. Eventos para Dominios de Aplicaciones

Evento	Descripción
Carga del ensamblado (a*ssemblyload*)	Este evento es disparado cada vez que el CLR carga un ensamblado en el dominio, y este recibe en un objeto *System.Reflection.Assembly* identificando el ensamblado cargado.
Descarga de dominio (*domainunload*)	Este evento es disparado antes de que el dominio sea descargado, y no es disparado cuando el proceso contenido en el dominio está terminando.
Salida de proceso (*processexit*)	Este evento es disparado antes de que un proceso termine, y es disparado solamente para el dominio default.
Excepción no manejada (*unhandledexeption*)	Este evento es disparado cuando un error no manejado ocurre en el dominio.
Resolución de ensamblado (*assemblyresolve*)	Este evento es disparado cuando el CLR no puede localizar un ensamblado requerido por el dominio. El manejador recibe una cadena de caracteres que identifica el nombre del ensamblado perdido.
Resolución de recurso (*resourceresolve*)	Este evento es disparado cuando el CLR no puede localizar un recurso requerido por el dominio. El manejador recibe una cadena de caracteres que identifica el nombre del recurso perdido.
Resolución de tipo (*typeresolve*)	Este evento es disparado cuando el CLR no puede localizar un tipo requerido. El manejador recibe una cadena de caracteres que identifica el nombre del tipo perdido, y el CLR puede decir que tipo retornará una referencia a un objeto de este tipo.

2.3 El lenguaje común de ejecución CLR

El lenguaje común de ejecución (*Common Language Runtime*) provee un entorno para el manejo de código ejecutable, para el código destinado al .Net Framework. El código manejado puede tomar el manejo de memoria, así como el manejo de la seguridad, la verificación del código, la compilación, y algunos otros servicios del sistema.

Los códigos manejados son validados variantemente según el grado de confianza, dependiendo de un número de factores que incluyen su origen (tal como el Internet, la red empresarial, o la computadora local). Estos componentes manejados podrían o no ser capaces de ejecutar operaciones de acceso a archivos, operaciones de acceso al registro u otras funciones sensibles, esto si se son usados en la misma aplicación activa.

El CLR fortalece la seguridad habilitando usuarios, asimismo fortalece el código para implementar una estricta verificación de tipos, en la infraestructura llamada el Sistema Común de Tipos (CTS). Este CTS se asegura que todos los códigos manejados sean descritos por si mismos, el código manejado puede consumirse con otras clases, tipos y objetos manejados, mientras estrictamente se fortalece la fidelidad de tipos y la seguridad de estos. Además, el entorno manejado por el CLR asegura que los tipos más comunes de software publicados son solventados o erradicados completamente. Los lenguajes que compilan para el .Net CLR hacen que las características de este, estén disponible para el código existente escrito en estos lenguajes, hasta ahora es bastante fácil el proceso de migración para las aplicaciones existentes. Aunque el CLR es diseñado para el software del futuro, este también soporta el de hoy en día, y los otros más antiguos.

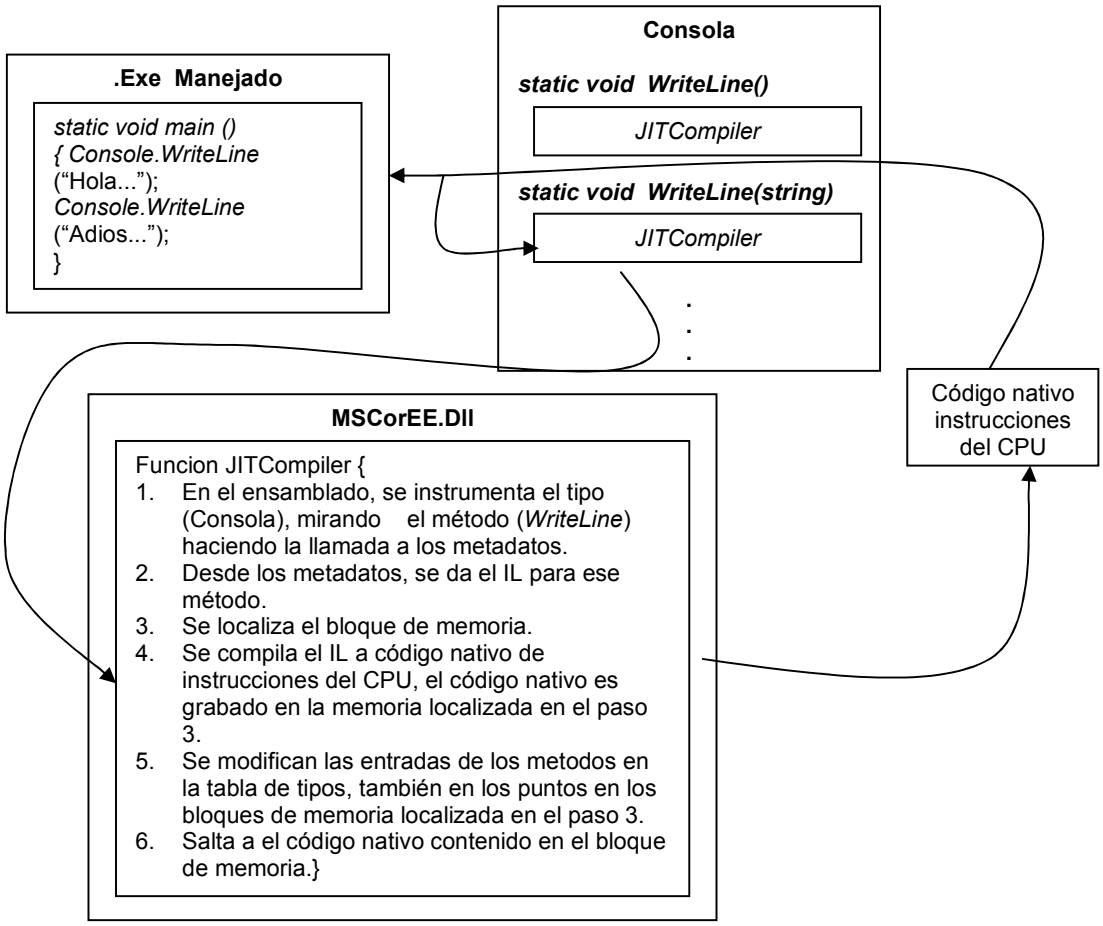
Cada uno de los ensamblados que son construidos puede ser una aplicación ejecutable o un Dll que contenga tipos o componentes, para ser usado por una aplicación ejecutable, el CLR es responsable de manejar el código ejecutable contenido en estos ensamblados, siendo lo que el .Net Framework instala el equipo anfitrión, ya que Microsoft ha creado un paquete de redistribución que se puede instalar libremente en los equipos clientes de las aplicaciones. Eventualmente el .Net *Framework* será empacada para futuras versiones de Windows, por lo que realmente se pude decir que es una especie de actualización para el sistema operativo, pero que a diferencia de los *Service Pack*, no lo afecta por completo, ya que implementa una forma nueva de utilización de aplicaciones, pero sin afectar las otras ya existentes.

Cuando se construyen aplicación en un ensamblado Exe, el compilador conecta alguna información especial en el resultado del encabezado del archivo PE, y cuando un Exe es invocado, esta información provoca que el CLR sea cargado e inicializado, posteriormente el CLR localiza el método inicial de la aplicación y permite que esta inicie su ejecución. De forma similar las aplicaciones no manejadas llaman a las librerías para cargar y manejar el ensamblado, los Dlls buscan la función de inicio sabiéndolo para cargar en el CLR en orden para procesar el código contenido dentro del ensamblado.

El CLR esta diseñado para fortalecer la ejecución de programas. Una característica de compilación llamada **justo a tiempo** (JIT *Just in Time*), habilita a todo el código manejado a correr en el código nativo de lenguaje de máquina del sistema en el cual se está ejecutando, siendo esto una característica muy importante, ya que los PE cuando se ejecutan por el CLR, son convertidos una única vez por el JIT, haciendo que la primera carga de los programas o de un nuevo componente sea un poco lenta, pero permite que a lo largo de la ejecución el funcionamiento sea mucho más optimo.

Justo antes de que el método principal sea ejecutado en un programa, el CLR detecta todos los tipos que son referenciados por el código principal. Esto causa que el CLR localice una estructura interna de datos que es usada para manejar el acceso a tipos referenciados. Los métodos principales referencian a un tipo simple, causando que el CLR localice una estructura simple interna. Esta estructura interna de datos contiene una entrada para cada método definido por el tipo. Cada entrada refugia la dirección donde el instrumento del método puede estar establecido. Cuando se inicializa esta estructura el CLR agrupa cada entrada a una interna, sin documentar funciones contenidas dentro del CLR por si mismo, estas funciones son llamadas *JITCompiler*.

Figura 12. Llamado a un método por primera vez (*JITCompiler*)



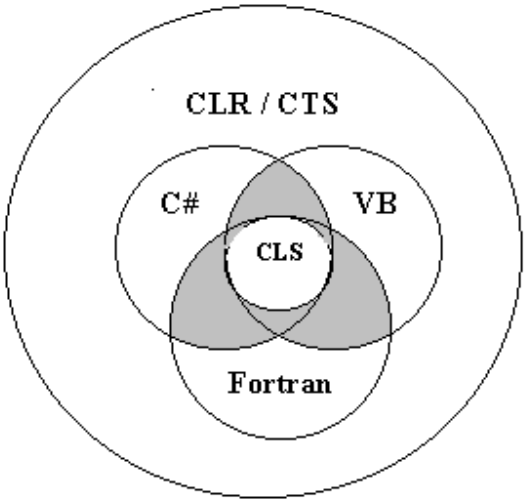
Cuando el procedimiento principal hace la primera llamada, la función *JITCompiler* es llamada, y esta es responsable por compilar los métodos en código IL hacia el código nativo de la máquina en la cual se está ejecutando el programa. Para entender la funcionalidad del CLR, es necesario aclarar que en realidad este es únicamente un interprete de un lenguaje único, en el cual nos enfocaremos en el siguiente capítulo, siendo el MSIL o lenguaje intermedio, pero lo más importante en cuando al .Net es que desde sus inicios fue liberado un grupo de estándares que permitirían realizar compiladores de cualquier lenguaje que pudiera acoplarse a clases y llevarlo a este MSIL, en lugar de hacerlo a instrucciones de un CPU específico, para que el código generado funcionara debía de cumplir con los estándares definidos pro Microsoft, siendo estos las **especificaciones del lenguaje común (c*ommon language specification*)**.

El COM permite objetos creados en diferentes objetos para comunicarse uno a otro. Por otro lado, el CLR ahora integra todos los lenguajes y permite objetos creados en un lenguaje para ser tratados como iguales por el código escrito en un lenguaje completamente diferente. Esta integración es posible porque los estándares mencionados, autodescripción de tipos de información (metadatos), y un entorno común de ejecución.

Mientras esta integración de lenguajes es una meta fantástica, la verdad del material es que los lenguajes de programación son muy diferentes desde uno a otro, sin ir tan lejos algunos lenguajes validan sensibilidad de caracteres y otros no, o tienen enteros sin signo y otros no, etc. Si se intenta crear un tipo que sea fácilmente accesado desde otro lenguaje de programación, se necesita el uso de características del lenguaje, garantizando que serán disponibles en todos los lenguajes.

El CLR/CTS (Lenguaje Común de Ejecución / Sistema Común de Tipos) soportan un lote grande de características de las cuales un subgrupo son definidas por el CLS, así si se cuida la interoperabilidad de los lenguajes, pudiéndose desarrollar tipos muy ricos limitados solamente por este grupo de características. Específicamente el CTS define reglas que externalizan tipos visibles y métodos que podrían adherirse si ellos son accesibles desde un compilador CLR. Es importante notar que las reglas del CLS no aplican al código que es accesible solamente entre ensamblados definidos.

Figura 13. Las especificaciones del lenguaje común



2.4 Las bibliotecas de clases del .Net *Framework*

El .Net *Framework*, incluye clases, interfaces y valores de tipos que facilita y optimiza el proceso de desarrollo y provee acceso a las funciones del sistema, esto para facilitar la interoperabilidad entre los lenguajes. El .Net incluye tipos que ejecutan las siguientes funciones:

- Representación de tipos de datos base y excepciones.
- Encapsulamiento de estructuras de datos.
- Ejecución de entrada / salida.
- Acceso a información acerca de la carga de tipos.
- Invocar chequeos de seguridad del .Net *Framework*.
- Provee acceso a datos, rico GUI del lado del cliente, y controles del lado del servidor.

Los tipos del .Net *Framework* usados una sintaxis de esquemas de nombres que contiene una herencia. Esta técnica agrupa tipos relacionados dentro de espacios de nombres, así ellos pueden ser buscados y referenciados de forma fácil. La primera parte del nombre completo es el nombre del espacio, y la última parte es el nombre del tipo, por ejemplo en el *System*, *Collections.ArrayList*, el tipo utilizado es *ArrayList* y este está contenido dentro del espacio de nombre *System.Collections*. Este esquema de nombres hace fácil para las bibliotecas desarrolladas extender el .Net *Framework* al crear grupos heredados de tipos y nombres con una consistencia informativa.

2.5 Manejo automático de memoria

Cualquier programa utiliza recursos, siendo archivos, memoria, espacios de pantalla, conexiones de red, recursos de bases de datos, etc., y uno de los factores en un entorno orientado a objetos, es que cualquier tipo identifica algún recurso disponible para uso de los programadores. Los siguientes pasos son requeridos para accesar a recursos:

- Localizar la memoria para el tipo que representará el recurso para llamar a las instrucciones ***newobj*** del IL, la cual se emite cuando se usa el operador ***new***, en lenguajes como C#, Visual Basic, y otros.
- Inicializar la memoria para configurar es estado inicial de los recursos y para hacer usables los recursos. Los constructores de tipos son responsables de la configuración de este estado inicial.
- Usar los recursos por acceso a los miembros de los tipos (repitiéndose como sea necesario).
- Rasgar bajo el estado de un recurso para limpiarlo.
- Liberar memoria. **El colector de desperdicios (*Garbage Collector*)** es únicamente para este paso.

Este aparentemente simple paradigma es uno de los mayores orígenes de los errores de programación ya que muchas veces se olvidan el último paso, la liberación de memoria, o se intenta accesar a la memoria cuando esta ya ha sido liberada. Estos dos errores son peor que la mayoría, porque usualmente no se puede predecir las consecuencias de estos problemas. Pero estos errores causan agujeros en los recursos, así como la corrupción de objetos, haciendo que el nivel de funcionamiento de la aplicación sea impredecible, siendo de este paradigma de donde surge un manejo automático proporcionado por el .Net *Framework*, el **colector de desperdicios (*Garbage Colector*)**.

La administración de los recursos propios es muy difícil y bastante tediosa y esto distrae a los desarrolladores de los problemas reales de una aplicación, ya que estarán intentando solventar problemas de manejo de recursos, al existir el colector de desperdicios, esto se facilitará, ya que completamente perdonará a los desarrolladores por el descontrol en el manejo de memoria, ya que una vez utilizada la memoria, este sabrá cuando liberarla. El colector no sabe cualquier cosa sobre los recursos representados por el tipo en memoria, por lo cual este no puede saber como ejecutar la limpieza. Para dar a los recursos una limpieza apropiada, los desarrolladores deberían escribir código que sepa como limpiar adecuadamente el recurso. Los desarrolladores escriben código en los métodos ***Finalize, Dispose*** y ***Close***. Sin embargo el GC puede ofrecer alguna asistencia sobre esto, permitiendo a los desarrolladores hacer la limpieza bajo muchas circunstancias.

Cuando un proceso es inicializado, el CLR reserva una región de contingencia de espacios de direcciones que inicialmente contiene un almacenaje de fondo. Este espacio de direccionamiento es manejado en la memoria, manteniendo aquí un puntero, el cual puede llamar a la función ***NextObjPtr***, que indica donde está el próximo objeto, inicialmente es configurado para la dirección base de la región de espacios de direcciones. El ***newObj*** del IL provoca que el CLR ejecute los siguientes pasos:

- Calcula el número de *bytes* requerido por los tipos de los campos.
- Agrega los *bytes* requeridos para los gastos generales del objeto. Cada objeto tiene dos campos de gastos, un puntero a tabla de métodos y uno a un índice de bloque de sincronización (*SyncBlockIndex*). Sobre un sistema de 32 bits, cada uno de los campos requiere 32 *bits*, agregando 8 más para cada objeto, mientras que en un sistema de 64 bits se agregan 16 más por cada objeto.

- El CLR chequea que los *bytes* requeridos para localizar el objeto son disponibles en la región reservada. Si los objetos aptos, estos son localizados en los punteros de direcciones dadas por el ***NextObjPtr***. El constructor del tipo es llamada, y la instrucción ***newObj*** regresa la dirección del nuevo objeto. Sola antes de que la dirección sea retornada, ***NextObjPtr*** es avanzado pasando el objeto e indica la dirección donde el próximo objeto será ubicado en la memoria.

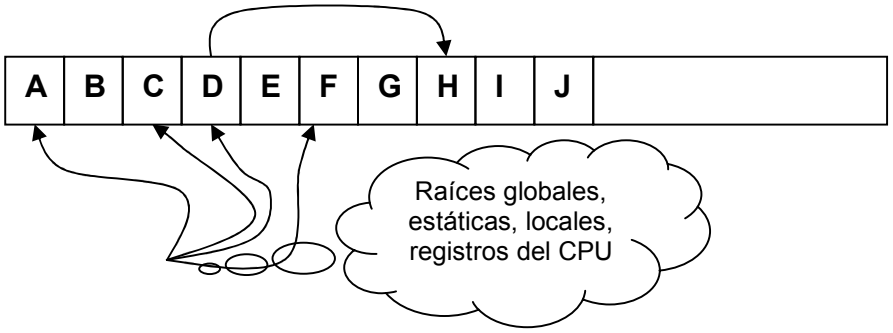
2.5.1 Algoritmo del colector de desperdicios

El colector cheques para ver si los objetos en la memoria son o no utilizados por alguna aplicación. Si tales objetos existen, la memoria usada para estos objetos puede ser reclamada. Y para verificar esta utilización de objetos, el colector se basa en su algoritmo de ejecución. Cada aplicación tiene una raíz, y una raíz simple es almacenada en una locación de memoria referenciada a través de un puntero. Este puntero referencia a un objeto en la memoria manejada o en un grupo de nulos. Por ejemplo, todos los globales o referencias estáticas de tipos son consideradas raíces, asimismo las referencias a variables locales o parámetros variables o pilas. Finalmente dentro de un método, un registro de CPU que referencia a un tipo de objeto referenciado está casi considerado como una raíz.

Cuando el JIT compila los métodos en IL, este produce código nativo para el CPU, asimismo este también crea una tabla interna. Lógicamente, cada entrada en esta tabla indica el rango de bytes configurados para los métodos nativos en instrucciones del CPU, y para cada rango, un grupo de direcciones de memoria o registros de CPU que contienen las raices.

Cuando el colector empieza a ejecutarse, asume que los objetos en la memoria son desperdicios, asume que ninguno es una raíz referenciada por una aplicación o por cualquier objeto en la memoria. El colector arranca recorriendo las raices, construyendo un gráfico de todos los objetos obtenidos. El colector puede localizar una variable global que es un puntero para un objeto en la memoria. La figura que se presenta a continuación se pueden observar varios objetos localizados en memoria donde la raíz de la aplicación refiere directamente a los objetos A,C,D y al F. Con esto el colector inicia su gráfico y cuando agrega el objeto D, el colector es notificado que este referencia al objeto H y también lo agrega al gráfico. El colector continúa posteriormente su recorrido dentro de la memoria obteniendo los objetos recursivamente.

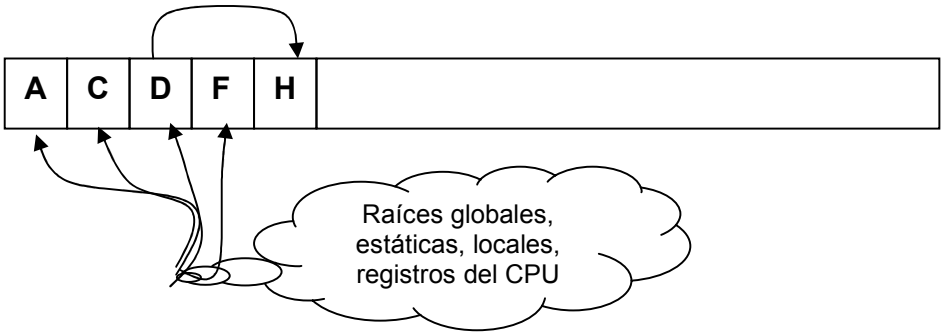
Figura 14. Manejo de la memoria antes del colector de desperdicios



Una vez la parte de la graficación de los objetos es finalizada, este chequea la próxima raíz y recorre los objetos nuevamente. Como el colector recorre de objeto en objeto, si este prueba agregar un objeto a la grafica que esta previamente agregado, este podrá parar de recorrer hacia abajo esta ruta. Este comportamiento sirve para dos propósitos, que el nivel de ejecución sea engrandecido significativamente, porque el colector no recorre dentro de un grupo de objetos más de una vez, y segundo es que los ciclos infinitos son prevenidos si se tiene cualquier conexión circular en una lista de objetos.

Una vez que todas lar rutas son chequeadas, la gráfica del colector contiene el grupo de todos los objetos que son obtenidos como raíces de aplicaciones; cualquier objeto que no esté en el gráfico no será accesible para una aplicación y son por lo tanto desperdicios. El colector ahora atraviesa la memoria linealmente viendo para el bloque de contingencias de los objetos desperdicio (ahora considerados espacio libre). Si los bloques encontrados son pequeños, el colector deja el bloque solo. Pero si el bloque de contingencia es grande, el colector cambia el objeto no desperdicio hacia debajo de la memoria, esto para compactar la memoria. Naturalmente, moviendo los objetos en memoria invalida todos los punteros a este objeto. El colector podrá modificar las raíces de las aplicaciones, así que el puntero raíz en el objeto direccione a la nueva localización. En conjunto con esto cualquier objeto que contiene un puntero a otro objeto, el colector es responsable de corregir ese puntero para que sea bueno. Después de que la memoria es compactada, el manejo del puntero ***NextObjPtr***, es el punto solo después del último objeto no desperdicio.

Figura 15. Manejo de la memoria después del colector de desperdicios



Como se puede visualizar en el gráfico anterior, un colector genera un considerable golpe en el nivel de funcionamiento de las aplicaciones, en las cuales el mayor problema de bajas se da a causa del uso mal administrado de la memoria. Finalmente el colector de desperdicios del CLR ofrece alguna optimización que mejora grandemente el nivel de ejecución del colector.

Un tipo que envuelve un recurso no manejado, tal como un archivo, una conexión a red, *sockets, mutex*, etc. soportan una **finalización**, la cual permite que un recurso sea limpiado por si mismo, cuando este sea colectado. Cuando el colector de desperdicios determina que un objeto es desperdicio, este llama el método de finalización del objeto, que generalmente es llamado ***Finalize***. Este método es usualmente implementado en llamadas como ***CloseHandle***, pasando el manejo de un recurso no manejado. Es por eso que ***FileStream*** define un ***Finalize*** para cada método, para cualquier objeto de estos es garantizado que los recursos manejados serán libres cuando el manejador de objetos es liberado. Si un tipo que envuelve un recurso no manejado falla al definir un método de finalización, el recurso no manejado no será cerrado y será un recurso agujero que existe hasta que el proceso termine, tal como los punteros, momento en el cual el sistema operativo reclamará los recursos no manejados.

3. EL ENSAMBLADOR DEL LENGUAJE INTERMEDIO

3.1 Fundamentos generales del ensamblador

El CLR es de muchos aspectos del concepto del .Net, pero lo principal es que este es su núcleo, razón por la cual la estructura general del .Net y sus componentes, es muy importante. Esto provee un capa de operación entre las aplicaciones y el sistema operativo. En un principio el CLR es similar a un lenguaje interpretado como *Gbasic* o como la máquina virtual de Java, pero en realidad no lo es, ya que no interpreta la información como los mencionados, sino realiza una compilación a lenguaje de CPU luego de la primera corrida del archivo PE, ya que ejecuta el *JITCompiler*.

Las aplicaciones .Net generadas por un compilador orientado a .Net, son representadas en una abstracta forma intermedia, independiente del lenguaje original de programación, y también independiente de la máquina destino de ejecución, así como su sistema operativo. Por esta representación, las aplicaciones .Net escriben un lenguaje diferente que puede interoperar fielmente, no solo en el nivel de llamadas de cada una de las funciones sino también al nivel de herencia de clases. El aceptar diferentes lenguajes de programación, debe de existir un grupo de reglas bien establecidas para las aplicaciones que sean generadas y ejecutadas en forma correcta. Este grupo de reglas garantizan la interoperabilidad de las aplicaciones .Net, siendo esto lo conocido como las especificaciones del lenguaje común CLS, que fue explicado en el capítulo anterior. Si existe una aplicación que no es compilada bajo el esquema CLS, será funcional en el sistema operativo, pero no podrá ser garantizada su interoperabilidad con aplicaciones en todos los niveles.

El código manejado representa la funcionalidad de los métodos de las aplicaciones codificadas en una forma binaria conocida como **lenguaje intermedio Microsoft MSIL**, o **lenguaje intermedio común CIL**, siendo este tipo de código el manejado por el CLR, y este manejo incluye tres actividades principales, que son los **tipos de controles**, el **manejo de excepciones** y el **colector de desperdicios**, encargándose el primero de la verificación y conversión de los tipos durante la ejecución, el segundo tiene una funcionalidad de excepciones no manejadas, siendo esto ejecutado por el CLR y no por el sistema operativo, y finalmente el colector que como ya se mencionó en el capítulo anterior se encarga de manejar en forma automática la eliminación de objetos que no están en uso.

Un programa ejecutable de código manejado puede ser precompilado desde el IL hacia el código nativo del CPU, por lo cual se puede hacer esto cuando el ejecutable esta esperando a ser ejecutado repetidamente desde un disco local, para evitar el consumo de tiempo de la compilación JIT. En este caso el código precompilado es grabado en un disco de almacenamiento, y cada vez que el ejecutable es invocado, la versión precompilada de código nativo es usada en lugar de la versión original en IL. El archivo original, también tiene que estar presente porque la versión precompilada no contiene los metadatos.

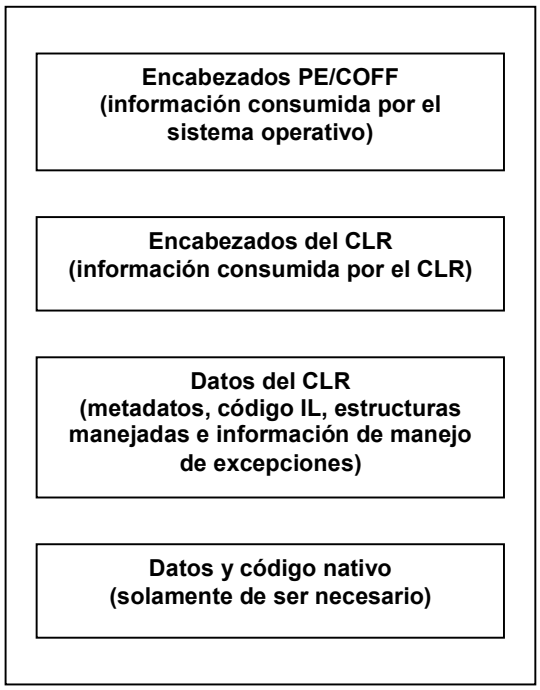
Existen dos cosas importantes en el manejo de código IL, y la primera de ellas es el saber acerca del mapeo de los campos y los tipos de valores como una especie de bodega de información, pero no necesitaremos emplear eso continuamente. Si un método acepta como primer parámetro o argumento una cadena de caracteres, este podrá solamente aceptar cadenas como el segundo argumento también. La segunda cosa es que podemos usar ciertos accesos directo en las instrucciones del IL.

En el IL existen dos formas de direccionamiento a las instrucciones IL, siendo la forma larga y la forma corta, se basan en la cantidad de *bytes* del salto para la búsqueda de las mismas, sin embargo los compiladores de lenguajes de alto nivel que generan código IL, automáticamente estiman el rango de llamadas de las instrucciones IL, y cambian si una forma larga o una forma corta de la instrucción puede ser usada en cada caso particular. El compilador ILAsm, utiliza como default las llamadas largas, sin embargo si se especifica si la instrucción es larga o corta, el compilador toma el valor al momento de enfrentarse con la compilación, pero si se especifica una instrucción corta y el lugar destino está en un rango largo, el compilador del IL podrá diagnosticar dicho error.

3.1.1 Manejo de estructura de archivos ejecutables

Ya se ha mencionado el concepto de los archivos portátiles ejecutables, y sabiendo como un módulo es manejado y ejecutado en el entorno del CLR, es también necesario saber el formato de un módulo manejado basado sobre los estándares del formato de archivos de objetos comunes COFF y sus extensiones para los PE. Así formalmente un módulo manejado es un archivo PE/COFF, con características adicionales que identifican esto como un ejecutable manejable. Porque el formato de archivos de módulos manejados conformado para el estándar Windows PE/COFF, el sistema operativo trata el manejo de módulos como un ejecutable. Y las extensiones, que es información de especificaciones para el CLR permite al ejecutor inmediatamente tomar control sobre el módulo de ejecución, tan pronto como el sistema operativo lo invocara.

Figura 16. Estructura de un archivo manejado PE/COFF



Las definiciones comunes del manejo de la estructura de archivos PE son:

- **Puntero de archivos**, es la localización de un ítem dentro del archivo mismo, antes de que este sea procesado por la carga. Esta localización es una posición dentro de archivo como este está almacenado en el disco.
- **Dirección virtual relativa RVA**, que es la dirección de un ítem una vez ha sido cargado en la memoria, con la dirección base del archivo de imágen substraído desde este, en otras palabra es la ubicación de un ítem dentro del archivo imágen cargado a memoria.

- **Dirección virtual VA,** es lo mismo que el RVA excepto que la dirección base del archivo de imagen no es substraída. La dirección es referenciada como virtual, porque el sistema operativo crea un espacio de direcciones virtuales distinta para cada proceso, independientemente de la memoria física. Para casi todos los propósitos, una dirección virtual puede ser considerada simplemente como una dirección. Una dirección virtual no es predecible como un RVA porque la carga forzada no carga la imágen en esta ubicación preferida, si existe un conflicto con el archivo de imágen ya cargada.
- **Sección**, que es la unidad básica de un código o dato dentro de un archivo PE/COFF. Además para la sección de código y datos, un archivo de imagen puede contener una numerosa cantidad de secciones, tales como .tls (hilera de almacenamiento local) o .reloc (relocalización), que tienen un propósito especial. Asimismo toda la materia prima de datos en una sección puede ser cargada contiguamente.

3.1.2 Organización de tablas de metadatos

Como ya se ha mencionado en capítulos anteriores, los metadatos son definiciones de datos que describen los datos, siendo esto una definición general, sin embargo esto es muy importante. En el contexto del CLR, los metadados son el medio de descripción de todos los ítems que son declarados o referenciados en un módulo. Porque el modelo de programación para el CLR es una herencia del modelo orientado a objetos, por lo que el ítem representado en los metadatos es una clase y sus miembros, con sus respectivos atributos, propiedades, métodos y relaciones. Lógicamente, los metadatos son representados como un grupo de nombres, cada uno representando una categoría de metadatos. Estos grupos son divididos en dos tipos, los metadatos *Heap* y las tablas de metadatos.

Un metadato *Heap* es un almacenamiento de estructura virtual, reteniendo una secuencia de contingencia de ítems. Estos son usados en los metadatos para almacenar cadenas de caracteres y objetos binarios, y estos tienen tres ventajas principales:

- ***Heap* de cadenas de caracteres**. Este tipo de heap contiene un carácter de terminación codificada en UTF-8. Las cadenas están seguidas una de otra inmediatamente. Porque el primer byte del heap es siempre es 0, la primera cadena en el heap esta vacía. El último *byte* del *heap* podría ser 0 también.
- ***Heap* GUID**, es un tipo que contiene 16 bytes de objetos binarios, inmediatamente seguido de otro, porque el tamaño del objeto binario es corregido, la longitud de los parámetros o los terminadores no son necesarios.
- ***Heap* gota**, es un tipo que contiene objetos binarios de un tamaño arbitrario, cada objeto es precedido por esta longitud (en forma comprimida). Los objetos binarios son alineados en límites de 4 *bytes*. La fórmula de compresión de longitud es bastante simple. Si la longitud (la cual es un entero sin signo) es 0x7F o menor, esto es representado como 1 *byte*; si la longitud es más grande que los 0x7F pero no mayor que 0x3FFF, esto es representado como un entero sin signos de 2 *bytes* con el mayor bit configurado. De otra manera, esto es representado como un entero sin signo de 4 *bytes* con los dos *bits* mayores configurados.

Tabla V. Compresión de valores de metadatos *heap*

Rango de Valores	Tamaño Comprimido	Valor Comprimido
0–0x7F	1 byte	<Valor>
0x80–0x3FFF	2 bytes	0x8000 | <Valor>
0x4000–0x1FFFFFFF	4 bytes	0xC0000000 | <Valor>

La fórmula de compresión de datos es utilizada en los metadatos. Por supuesto trabajan solo por números no excedentes de 0x1FFFFFFF (536,870,911), pero esta limitación no es un problema porque la compresión es usualmente aplicada para tales valores como longitudes y contadores.

El encabezado general de este tipo de metadatos consiste en una firma de almacenamiento y el almacenamiento de encabezados.

Tabla VI. Estructura de encabezado de metadatos

Tipo	Campo	Descripción
dword	*Lsignature*	Firma para metadatos físicos, actualmente 0x424A5342.
word	*imajorversion*	Versión mayor (1 para las primeras liberaciones del CLR)
word	*iminorversion*	Versión menor (0 para las primeras liberaciones del CLR)
dword	*Iextradata*	Reservado, inicialmente configurado a 0.
dword	*Ilength*	Longitud de la versión de cadena de caracteres.
byte[ ]	*iversionstring*	Versión de cadena de caracteres.

El almacenamiento de encabezado está seguido del almacenamiento de la firma, alineándolo sobre un límite de 4 bytes. Esta estructura es simple:

Tabla VII. Estructura de almacenamiento de firmas de metadatos

Tipo	Campo	Descripción
byte	*fflags*	Reservado; configurado inicialmente a 0.
byte		Relleno.
word	*streams*	Número de steams.

El almacenamiento de encabezado está seguido por un arreglo de encabezados de los streams. La estructura es la siguiente:

Tabla VIII. Estructura de encabezados de streams en metadatos

Tipo	Campo	Descripción
dword	*ioffset*	Compensación en los archivos para este stream.
dword	*isize*	Tamaño del *streams* en *bytes*.
char[16]	*rcname*	Nombre del *stream*; un terminador de cadenas ANSI, que no rebase la longitud de los siete caracteres.

Seis streams pueden ser representados en los metadatos:

1. #*Strings*, una cadena conteniendo el nombre de los ítems de metadatos (nombres de clases, nombres de métodos, nombres de campos, etc.). Los streams no contienen constantes literales definidas o referenciadas en los métodos de los metadatos.
2. #*Blob*, este contiene objetos binarios de metadatos internos, tales como valores default. Este no contiene objetos binarios definidos en los métodos de los módulos.
3. #GUID A GUID, contiene toda clase de identificadores globales únicos.

4. #US, un heap de tipo gota contiene cadenas definidas por el usuario. Este stream contiene cadenas definidas en el código de usuario. Las cadenas son mantenidas en codificación Unicode. Este stream es la característica más interesante, es que las cadenas de usuario pueden ser explícitamente direccionadas por el código IL.
5. #~, es un *stream* de metadatos comprimido (optimizado). Este contiene un sistema optimizado de tablas de metadatos.
6. #-, es un *stream* de metadatos sin compresión (no optimizado). Este contiene un sistema no optimizado de tablas, incluyendo los candados intermedios de tablas (punteros a tablas).

Los *streams* #~ y #-, son mutuamente excluyentes, es decir que la estructura de los metadatos de los módulos pueden estar optimizados o no, pero no ambos al mismo tiempo. Si los ítems son almacenados en un *stream*, estos son ausentes o nulos, y los campos *iSteams* del almacenamiento son correspondientemente reducidos. En un mínimo de tres *streams* son garantizados a estar presentes, un stream de metadatos (#~ o #-), un *stream* de caracteres (#*String*) y uno GUID (#GUID). Los ítems de metadatos pueden ser presentados en una configuración mínima en los eventos del módulo más trivial, estos ítems de metadatos tienen nombres y GUIDs.

Figura 17. Referenciación de *streams* por clientes externos

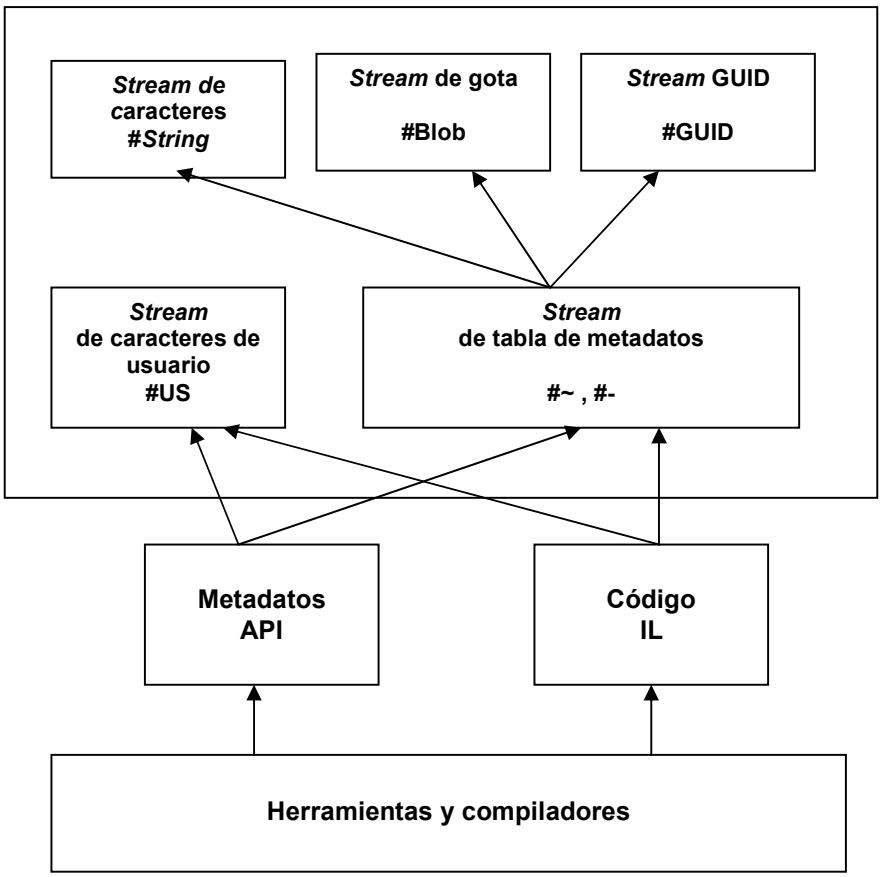


Tabla IX. Encabezados para *stream* de metadatos #~ y #-

Tamaño	Campo	Descripción
4 *bytes*	*reserverd*	Reservado; configurado inicialmente en 0.
1 *byte*	*major*	Versión mayor del esquema de tablas (1 para la primera liberación del CLR).
1 *byte*	*minor*	Versión menor del esquema de tablas (0 para la primera liberación del CLR).
1 *byte*	*heaps*	Banderas binarias indicando la compensación de tamaño para ser usado entre los *heaps*. Un entero de 4 *bytes* sin signo es indicador para 0x01 para un *heap* de tipo cadena, 0x02 para un *heap* GUID, y 0x04 para un *heap* de gota. Un *stream* puede también tener banderas especiales como 0x20, indicando que el steam contiene solo cambios hechos durante una edición y sesión continua, y una bandera 0x80, indicando que los metadatos pueden contener ítems marcados como eliminados.
1 *byte*	*rid*	Contador de bits del registro máximo en los índice para todas las tablas de los metadatos, calculándolo en tiempo de ejecución (durante la inicialización de los stream de metadatos).
8 *bytes*	*maskvalid*	Vector de *bits* de tablas presentes, cada *bit* representa una tabla.
8 *bytes*	*sorted*	Vector de *bits* de orden de tablas, cada *bit* representando una tabla respectiva.

Este encabezado es seguido por una secuencia enteros sin signo de 4 *bytes*, indicando el número de registros en cada tabla marcada, 1 en el vector de bits *MarkValid*. Como una base de datos, los metadatos tendrán un esquema, el cual es un sistema de descriptores de metadatos de tablas y columnas. Un esquema no es parte de los metadatos, ni es un atributo de un manejo de archivos PE.

Tabla X. Descriptores para tablas de metadatos

Tipo	Campo	Descripción
pointer	*pcoldefs*	Puntero a un arreglo de columnas de descriptores.
byte	*ccols*	Número de columnas en la tabla.
byte	*ikey*	Indice de las llaves de la columna.
word	*cbrec*	Tamaño de un registro en la tabla.

Los descriptores de columnas, para cada campo pColDefs de un puntero de descriptores de tablas, tiene la siguiente estructura:

Tabla XI. Estructura de descriptores de *PColDefs*

Tipo	Campo	Descripción
byte	*type*	Código de tipos de columnas.
byte	*ocolumn*	Compensación de la columna.
byte	*cbcolumn*	Tamaño de la columna en bytes.

El primer campo de una columna de descriptores, es especial, ya que el esquema de metadatos de la primera liberación del CLR identifica los códigos para los tipos de las columnas, como se presenta en la tabla XII.

Tabla XII. Codigos de tipos de columnas de descriptores (primer campo)

Rango	Descripción
0–63	La columna tiene el índice de registros RID, a otra tabla; el valor específico indica que tabla. El ancho de la columna es definido por el campo RID de los metadatos del encabezado del stream.
64–95	La columna tiene un código de referencia a otra tabla; el valor específico indica el tipo de señal codificado. Las señales son referenciadas acarriando los índices de ambos, de la tabla y del registro referenciado. Las tablas son direccionadas y el índice de los registros es definido por el valor de la señal codificada.
96	La columna tiene un entero con signo de 2 *bytes*.
97	La columna tiene un entero sin signo de 2 *bytes*.
98	La columna tiene un entero con signo de 4 *bytes*.
99	La columna tiene un entero sin signo de 4 *bytes*.
100	La columna tiene un entero sin signo de 1 *byte*.
101	La columna tiene una compensación en el *heap* de caracteres
102	La columna tiene una compensación en el *heap* GUID (#GUID)
103	La columna tiene una compensación en el *heap* de gota (#Blob)

El esquema de metadatos define una gran cantidad de tablas definidas.

Modulo, el actual descriptor de módulos.

TypeRef, descriptores de referencias de clases.

TypeDef, descriptores de definiciones de clases o interfaces.

FieldPtr, una tabla de búsqueda de clases para campos, los cuales no existen en metadatos optimizados (#~ *stream*).

Fields, descriptores de definiciones de campos.

MethodPtr, una tabla de búsqueda de clases para métodos, los cuales no existen en metadatos optimizados (#~ *stream*).

Method, descriptores de definición de métodos.

ParamPtr, un tabla de búsqueda de métodos para parámetros, los cuales no existen en metadatos optimizados (#~ *stream*).

Param, los descriptores de definición de parámetros.

InterfaceImpl, descriptores de realización de interfaces.

MemeberRef, descriptores de referencias a miembros (campos o métodos).

Constant, **descriptores de valores constantes que mapean los valores default almacenados en los steams tipo #Bob para los respectivos campos, parámetros y propiedades.**

CustomAttribute, descriptores de atributos personalizados.

FieldMarshal, descriptores de administración de campos o parámetros, para la interoperación entre los manejados y los que no.

DeclSecurity, descriptores de seguridad.

ClassLayout, descriptores de clases que tienen información acerca de cómo tienen que ponerse a cargar las clases externas respectivas.

FieldLayout, descriptores de campos que especifican la compensación o secuencia de los campos individuales.

StandAloneSig, descriptores de firmas independientes. Las firmas para ser usados en dos capacidades, como firmas compuestas o variables locales de métodos, y como parámetros de las llamadas indirectas a instrucciones IL.

EventMap, una tabla de mapeo de clases para eventos. Esto no es una tabla de búsqueda intermedia, y esto existe en los metadatos optimizados.

EventPtr, una tabla de búsqueda de eventos mapeados a eventos, los cuales no existen en metadatos optimizados.

Event, descriptores de eventos.

***PropertyMap*,** una tabla de mapeo de clases para propiedades. Esta no es una tabla de búsqueda intermedia, y existe en los metadatos optimizados.

PropertyPtr, es una tabla de búsqueda de propiedades mepeando a propiedades, lo cual no existe en metadatos optimizados.

Property, descriptores de propiedades.

MethodSemantics, descriptores semánticos de métodos que tiene información acerca de cual es asociado con una propiedad o evento específico.

MethodImpl, descriptores de implementación de métodos.

ModuleRef, descriptores de referencias de módulos.

TypeSpec, descriptores de especificaciones de tipos.

ImplMap, descriptores de mapas de implementación, usado para la invocación de plataforma (*P/Invoke*) interoperación de tipos de código manejado y no manejado.

FieldRVA, descriptores de mapeo de campos para datos.

ENCLog, descriptores de edición y logs continuos que tienen información de que cambios tienen que ser hechos para ítems de metadatos específicos durante la edición de memoria. Esta tabla no existe en los metadatos optimizados.

ENCMap, descriptores de edición y mapeo continuo. Esta tabla no existe en los metadatos optimizados.

Assembly, descriptor del ensamblado actual, el cual tendrá que presentarse solamente en el módulo principal de metadatos.

AssemblyRef, descriptores de referencia a ensamblados.

File, descriptores de archivos que contienen información de los archivos en el ensamblado actual.

ExportedType, descriptores de tipos exportados que contiene información acerca de clases pública exportadas por el ensamblado actual, son declarados en otro módulo de el ensamblado. Solo el módulo principal del ensamblado tendrá que acarrear esta tabla.

ManifiestResource, descriptores de los recursos manejados.

NestedClass, descriptores de clases anidadas que proveen mapeo a sus respectivas clases que los encapsulan.

3.1.3 Módulos y ensamblados

Como ya se ha mencionado antes, un ensamblado es básicamente una unidad de desarrollo, un bloque construido de aplicaciones manejadas. Los ensamblados son reutilizables, permitiendo diferentes aplicaciones para usar el mismo ensamblado, ya que estos acarrean su propia descripción de sus metadatos, incluyendo la información de versiones que permite el CLR para usar versiones específicas de un ensamblado para una aplicación particular.

3.1.3.1 Ensamblados compartidos y privados

Los ensamblados son clasificados como privados o compartidos, según su utilización, estructuralmente y funcionalmente, estas dos maneras de ensamblados son los mismos, pero son diferentes en como son llamados y distribuidos y en el nivel o versión de chequeo de ejecución para la carga. Un ensamblado privado es considerado parte de una aplicación particular. Un ensamblado privado es usualmente creado para el mismo autor como otros componentes específicos para una aplicación y esto es considerado para ser responsabilidad primariamente del autor. El nombre de un ensamblado privado puede ser únicamente utilizado dentro de la aplicación. Como un resultado de tal posición, los requerimientos llamados y versionados para ser ensamblados compartidos son llamadas de una forma mucho más estricta que los privados, ya que estos nombres deben de ser únicos globalmente.

La identificación de los ensamblados es provista por los nombres fortalecidos, los cuales utilizan llaves criptográficas públicas y privadas, que aseguran que el nombre sea único y para prevenir un nombre falso. Un nombre fortalecido también provee al cliente de un ensamblado compartido con información acerca de la identidad del ensamblado publicado. Si el CLR chequea la criptografía, el cliente puede asegurase que el ensamblado viene desde una publicación esperado, asumiendo que los encriptación privada de llaves no fue comprimida. Los ensamblados compartidos son distribuidos dentro del conocido caché global de ensamblados GAC, este almacena múltiples versiones de ensamblados compartidos lado a lado.

3.1.3.2 Dominios de aplicaciones como unidades lógicas

El sistema operativo y el CLR típicamente provee algo desde aislamiento ente aplicaciones corriendo en el sistema. Este aislamiento es necesariamente para asegurar que el código en ejecución, en una aplicación no puede afectar a otras aplicaciones. En los sistemas operativos modernos, este aislamiento es alcanzado para usar límites de procesos, donde un proceso, ocupando un único espacio de direcciones virtuales, corriendo exactamente una aplicación y alcanzando los recursos que están disponibles para el proceso a utilizar. La ejecución de código manejado tiene similares necesidades para aislamiento. Tal aislamiento puede ser provisto con bajo costo en aplicaciones manejadas, sin embargo, considerando que las aplicaciones manejadas corren bajo el control del CLR y son verificadas para ser del mismo tipo. El CLR permite múltiples aplicaciones para ser ejecutadas en un proceso simple del sistema operativo, usando un constructor llamado dominio de aplicación, para aislar las aplicaciones una de otra, por lo que se puede decir que un dominio equivale a un proceso en el sistema operativo.

Específicamente, el aislamiento en aplicaciones manejadas es el siguiente:

- Diferentes niveles de seguridad, pueden ser asignados para cada dominio de aplicaciones, dando al anfitrión un cambio para ejecutar las aplicaciones con variantes requerimientos de seguridad en un proceso.
- Las aplicaciones pueden ser independientemente detenidas y corridas linea por linea.
- El código que se esta ejecutando en una aplicación no puede accesar directamente al código de recursos desde otra aplicación.
- Defectos en una aplicación no pueden afectar a otra aplicación para botar el proceso entero.
- Cada aplicación tiene control sobre donde cargar el código, esta indicación viene desde que la versión de código es cargada. Adicionalmente, la información de configuración es recibida por la aplicación.

3.1.3.3 Los manifiestos de los ensamblados

Los metadatos describen a un ensamblado y este módulo es refenciado como un manifiesto, acarreando la siguiente información:

- **Identificación**, incluye un nombre textual, un número de versión, una cultura opcional si el ensamblado contiene localización de recursos manejados, y una llave pública opcional si el ensamblado tiene un nombre fortalecido. Esta información es definida en dos tablas de metadatos, los módulos y son ensamblados (en el modulo principal solamente).

- **Contenido**, incluyendo tipos y recursos manejados expuestos por este ensamblado, para uso externo y la localización de estos tipos y recursos. Las tablas que contienen esta información son ***ExportedType*** (en el módulo principal solamente) y ***ManifiestResource*.**
- **Dependencias**, incluye otros ensamblados externos, estos ensamblados son referenciados y en el caso de un ensamblado multimódulos, otros módulos del mismo ensamblado. Se puede buscar la información de dependencia en las tablas de metadatos ***AssemblyRef***, ***ModuleRef***, y ***File***.
- **Requerimientos de permisos**, específica al ensamblado como un entero. Más específicamente, este puede también ser definido por ciertos tipos o clases, así como métodos. Esta información de definida en la tabla ***DeclSecurity***.
- **Atributos personalizados**, específicamente para los componentes del manifiesto. Estos atributos proveen información adicional usada por compiladores y otras herramientas. El CLR define estos en la tabla ***CustomAttribute***.

La regla general en el ILAsm es “declarar, entonces referenciar”. Existe una secuencia de declaraciones recomendadas en la programación, con la declaración de manifiestos en las declaraciones en código fuente.

1. Declaraciones de ensamblados referenciados (*.assembly extern*), debido a los atributos personalizados. La referencia a los ensamblados puede conducir al empaquetamiento, debido a más atributos personalizados referenciados en este ensamblado.
2. Declaraciones de módulos referenciados (*.module extern*) nuevamente debido a los atributos personalizados.

3. Declaración de ensamblador *(.assembly*), esto debido a que el compilador del ILAsm, toma diferentes rutas en la compilación del ensamblado y de cualquier otro ensamblado, esto es mejor para conocer la ruta que tomará tan pronto como sea posible.
4. Declaraciones de archivos (*.file*), esto debido a las declaraciones referenciadas en la tabla **e*xportedtype*** y ***manifiestresource***.
5. Declaración de tipos exportados (*.class extern*), con Encapsulamiento de las declaraciones de tipos exportados precediendo las declaraciones anidadas de estos tipos.
6. Declaraciones de manifiestos de recursos (*.mresource*).

Una parte importante de los metadatos, es su contenido y debido a ello las reglas de validación de los mismos, ya que estas regla tienen impacto directo sobre la carga de las funciones ya que los chequeos respectivos son ejecutados en tiempo de ejecución. Otras reglas describen los metadatos "*well-formed*" o "bien formados"; estar violando una o varias reglas puede resultar en efectos bastante peculiares durante la ejecución de los programas, pero esto no representa una caída o brecha de riesgo en la seguridad, así la carga no ejecuta estos chequeos. Para buscar si uno de los metadatos en un módulo es individual, se puede correr la utilidad ***PEVerify***, usando la opción de validación de metadatos **/MD**. Alternativamente, se puede invocar al Desensamblador utilizando la opción **/ADV** o avanzada, eligiendo vistas (**v*iew***), Información de metadatos (***metaInfo***), lo cual esta construido dentro del CLR.

3.1.3.4 Reglas de validación de tablas de los ensamblados

- El contador de registros de la tabla puede ser no mayor que 1. Esto no es chequeado en tiempo de ejecución, porque la carga ignora todos los registros del ensamblado excepto el primero.
- La entrada de banderas debe tener bits configurados solo como los definidos en el ***CorAssemblyFlags***.
- La entrada local debe estar configuradas a 0 o referenciar a una cadena de caracteres no vacía, en el heap de caracteres que haga juego con una cultura de nombres conocida.
- Se puede obtener una lista de las culturas usando una llamada al método ***CultureInfo.GetCultures***, de la biblioteca de clases del *Framework*. Si la entrada local no está configurada a 0, la cadena referenciada no puede ser más larga que 1023 caracteres más el carácter terminador.
- La entrada nombre puede referencia a una cadena no vacía en el heap de caracteres. El nombre puede ser el nombre de archivo del módulo, excluyendo las extensiones y las rutas.
- La entrada de clave pública pueden ser configuradas a 0 o contener una compensación válida en el stream de gota #Blob.

3.1.3.5. Reglas de validación de tablas de los ensamblados referenciados

- La entrada de banderas pueden tener solo el mínimo significado de configuración de bits.
- La entrada de señal o clave pública puede ser configurada a 0 o contener una compensación válida en el stream de gota #Blob.
- La entrada local puede ser compensada con la misma regla de las tablas de los ensamblados.

- La tabla no puede tener registros duplicados simultáneamente compatibles en nombre, local y señal o clave pública, así como todas las entradas de versiones.
- **La entrada combes puede referenciar a una cadena no vacía. El nombre puede ser el mismo, que el del archivo del módulo principal.**

3.1.3.6 Reglas de validación de tablas de módulos

- El contador de registros de la tabla puede ser menor a 1.
- El contador de registros puede ser exactamente 1, esto no es un chequeo en tiempo de corrida porque la carga usa el primer registro en el módulo e ignora los otros.
- El entrada nombre puede hacer referencia a una cadena de caracteres no vacía, en el heap, cuya longitud no puede ser mayor a 511 caracteres más el terminador.
- La entrada Vid puede ser referenciada para un GUID que no sea cero, en el heap. El valor de esta entrada es generada automáticamente y no puede ser específicamente explicita en el ILAsm.

3.1.3.7 Reglas de validación de tablas de módulos referenciados

- **La entrada nombre puede hacer referencia a una cadena de caracteres no vacía, en el *heap*, cuya longitud no puede ser mayor a 511 caracteres más el terminador. El nombre puede ser el mismo que el del archivo del módulo principal.**

3.1.3.8 Reglas de validación de tablas de archivos

- La entrada de banderas pueden tener solo el mínimo significado de configuración de bits.
- **La entrada nombre puede hacer referencia a una cadena de caracteres no vacía, en el heap, cuya longitud no puede ser mayor a 511 caracteres más el terminador. El nombre puede ser el mismo que el del archivo del módulo principal.**
- **La cadena referenciada en la entrada nombre pude no hacer juego con S [N] [[C]*], donde:**
  - **S:= con │ Aux. │ Lat. │ Pan │ jul │ com.**
  - **N ::= 0..9**
  - **C ::= $ │ :**
- **La entrada de valor hash, puede tener una compensación válida en el stream de gota #Blob.**
- **La tabla no puede contener registros duplicados, cuyo nombre haga referencia a una cadena igual en contenido.**
- **La tabla no puede contener registros duplicados, cuyo nombre haga referencia a un nombre igual al del módulo.**

3.1.3.9 Reglas de validación de tablas de manifiestos de recursos

- La entrada de implementación puede ser configurada a 0 o puede tener una señal a un archivo o a un ensamblado referenciado.

- Si la entrada de implementación no tiene una señal a un ensamblado, la entrada de compensación puede tener una compensación válida entre los límites específicos del directorio de recursos de datos del CLR.
- La entrada de banderas podría tener cualquiera indicador 1 o 2, si es pública o privada respectivamente.
- La entrada de nombre puede referenciar a una cadena no vacía en el heap.
- **La tabla no puede contener registros duplicados cuyo nombre haga referencia a una cadena igual en contenido.**

3.1.3.10 Reglas de validación de tablas de tipos exportados

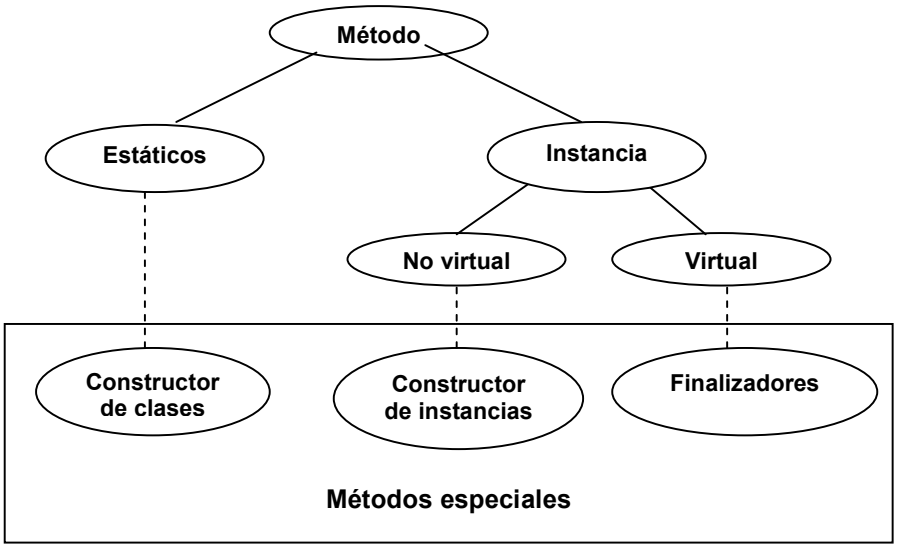
- El contador de registros de la tabla puede ser 0 si la tabla de ensamblados está vacía.
- El contador de registros de la tabla puede ser 0 si la tabla de archivos está vacía.
- La tabla no puede tener filas con nombres de tipos o espacios para nombres de tipos que sean iguales al nombre y al nombre del espacio respectivamente, de cualquier fila o tabla en la definición de tipos.
- La entra de bandera puede tener uno o más banderas visibles en la enumeración. Las banderas válidas están entre 0 y 7.
- La entrada de implementación puede tener una señal válida a un archivo o a un tipo exportado, así como puede tener una señal apuntando a un registro.
- Si la entrada de implementación tiene una señal a un tipo exportado, la entrada de bandera puede tener un valor de visibilidad entre 2 y 7.
- Si la entrada de implementación tiene una señal a archivo, la entrad de banderas puede tener un valor de visibilidad entre 0 y 1.
- La entrada de nombre de tipo puede referenciar a una cadena no vacía en el heap.

- La entrada de espacio de nombres puede ser configurada a 0 o referenciar una cadena no vacía en el *heap*.
- La longitud combinada de las cadenas referenciadas por los nombres de tipos y por los espacios de nombres no pueden exceder los 1023 caracteres.
- La tabla no puede contener registros duplicados cuya entrada de implementación tenga una señal a archivo, y cuya entrada de nombre de tipo y espacio de nombres de tipo refiera a una cadena con el mismo contenido.
- La tabla no puede contener registros duplicados cuya entrada de implementación tenga la misma señal a un tipo exportado, y cuya entrada de nombre de tipo refiera a una cadena con el mismo contenido.

3.1.4 Métodos

Los métodos como ya se ha mencionado, forman parte de las diversas clases que se pueden crear, y estos pueden ser clasificados como métodos estáticos, instancias y métodos virtuales. Los métodos estáticos son compartidos para todas las instancias de un tipo. Sello no requieren un puntero a una instancia, y no pueden ser accesados por instancias miembros menores que el puntero a instancia que lo provee explícitamente. Cuando un tipo es cargado los métodos estáticos son colocados en una tabla de tipos separada.

Figura 18. Clasificación de métodos



La firma de un método estático es la siguiente:

.metod public static void Bar (int32 I, float32 r)
{ ….. }

Los métodos que son instancias específicas tienen su declaración de firma de la siguiente forma:

.metod public instance void Bar (int32 I, float32 r)
{ ….. }

Se debe ser cuidadoso acerca del uso de la palabra clave ***instance***, en la especificación de la convención de llamadas a métodos, ya que cuando un método es definido, estas banderas (incluyendo las banderas estáticas) son específicamente explicitas. Debido a que esto en tiempo de definidos no es necesario para especificar la convención de llamadas a instancias, esto puede ser deducido, sin embargo es mejor que esta palabra clave sea especificada para métodos instancias, de lo contrario serán asumidos como estáticos.

Los métodos instancia son divididos en virtuales y no virtuales, identificándose por la presencia o ausencia de la bandera virtual. Los métodos virtuales de una clase son llamados a través de tablas de métodos virtuales (v-tables) de esta clase, lo cual agrega otro nivel indirecto a través de los instrumentos de llamadas. Los métodos virtuales pueden ser sobrescritos en clases derivadas por su propio método virtual de la misma forma y nombre. Si se tienen métodos no virtuales declarados en una clase, esto no puede declarar otro método no virtual con la misma firma en una clase derivada.

3.1.4.1 Atributos de los encabezados de métodos

El valor RVA de un registro método apunta a el cuerpo del método, y este consiste de un encabezado, el código en IL y un estructura opcional que es una tabla de manejo de excepciones (SEH). Existen dos tipos de encabezados, el **Tiny** y el **Fat**, donde el último es mucho más completo.

Tabla XIII. Estructura de encabezados de metadato

Entrada	Descripción
word	Los dos *bits* menores tienen el código del tipo de encabezado, los siguientes 10 tienen las banderas. Los cuatro bits mayores tienen el tamaño del encabezado en palabras dobles y puede estar configurado a 3. Actualmente las banderas son 0x2, los cual indica que las demás secciones son el código IL (que es una tabla SEH), y 0x4, los que indica que las variables locales pueden ser inicializadas.
word	***MaxStack***, que es la evaluación de la profundidad máxima de la pila en ranuras. El tamaño de la pila en IL no es preciso en *bytes*, pero si en ranuras, con cada ranura se tiene la habilidad de aceptar un *item* a pesar del tamaño de los *items*. El valor *default* es de 8 ranuras, y esto puede ser explícitamente en ILAsm por la directiva *.maxstack* <entero> usado dentro del alcance del método. Se debe ser cuidadoso acera de molestias para economizar el tamaño de los métodos para especificar el *.maxstack* como menor que el *default*.
dword	*CodeSize*, que es el tamaño del código IL en *bytes*.
dword	*LocalVarSigTok*, que es la señal de la firma de una variable local.

Figura 19. Estructura de encabezado *tiny*

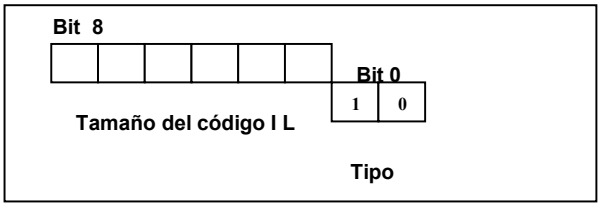
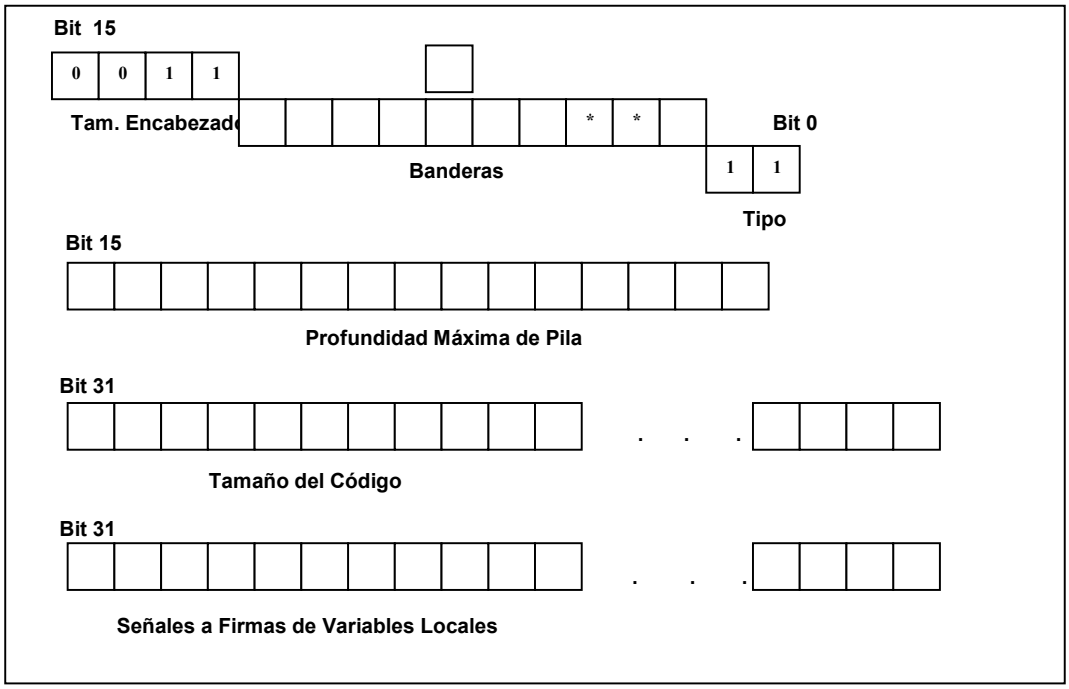


Figura 20. Estructura de encabezado *fat*



3.1.5 Variables locales

Las variables locales son los ítems de tipos de datos que son declarados dentro del alcance de los métodos y existe desde el momento que el método es llamado, hasta que este retorna. ILAsm permite que se asignen nombres a las variables y referenciarlas por su nombre, ya que las instrucciones IL direccionan las variables locales. Cuando el código fuente es compilado en modo de debugeo, los nombres de variables locales son almacenados en los programas de archivos de bases de datos (PDB) acompañando el módulo, y en este caso los nombres de variables locales pueden sobrevivir a lo largo de la ejecución. En general, estos nombres no son preservados debido a que las diferencias de los nombres de campos y parámetros de métodos no son parte de los metadatos. Todas las variables locales, no cuestiona cuando ellas son declaradas entre los límites de los métodos, desde una firma simple, manteniendo en las tablas de metadatos *StandAloneSig*.

3.1.6 Constructor de clases

Los constructores o inicializadores de clases , son métodos específicos para un tipo como un entero que corre después de que el tipo es cargado y antes de que los miembros del tipo sean accesados. Estos constructores son estáticos y tienen nombres especiales y sus respectivas banderas, asimismo no tienen ni parámetros ni retornan algún valor, y tiene el nombre **.cctor**, el cual en ILAsm es una palabra clave especial para esto. Solo es permitido un constructor de clases por tipo, y este no puede ser usado por la convención llamando argumentos variables.

Normalmente, los constructores nunca son llamados desde el código IL. Si un tipo tiene un constructor de clases, este constructor es ejecutado automáticamente después de que el tipo es cargado. Sin embargo un constructor de clases, como un método estático puede ser llamado explícitamente, y como resultado de esta llamada, los campos globales del tipo son reiniciados a sus valores de inicialización. Llamando al .cctor explícitamente no se realiza un recarga del tipo.

3.1.7 Constructor de instancias

Los constructores de instancias, son distintos a los constructores de clases, son específicamente para un instancia de un tipo y son utilizadas para inicializar tanto campos estáticos e instancias del tipo. Funcionalmente, los constructores de instancia en el IL son directamente análogos a los constructores de C++. Estos puede tener parámetros pero no retornan ningún valor, y también pueden tener un nombre especial y sus respectivas banderas, y este tiene el nombre **.ctor**, siendo esta también una palabra clave en ILAsm. Como los constructores de clases, los constructores de instancias no pueden usara la convención de llamadas a argumentos variables. Usualmente los constructores de instancias son llamados durante la ejecución de la instrucción ***newobj***, cuando un nuevo tipo de instancia es creado. Como en el caso de los constructores de clases, un constructor de instancias puede ser llamado explícitamente, llamando a la instancia del constructor este reiniciará los campos de una tipo pero no creará una instancia nueva. El único problema con llamar a constructores de clases o instancias explícitamente es que algunas veces el constructor incluye la instanciación de tipos.

3.2 Gramática del ILAsm

3.2.1 Instrucciones del lenguaje intermedio

A continuación se presentarán las instrucciones existentes en el lenguaje intermedio (Los detalles de la gramática son presentados en el Anexo C), pero para ello debemos de identificar inicialmente dos cosas:

Tabla XIV. Tipos de parámetros válidos en ILAsm

Tipo	Descripción
int8	Entero con signo de 1 *byte*
uint8	Entero sin signo de 1 *byte*
int32	Entero con signo de 4 *bytes*
uint32	Entero sin signo de 4 *bytes*
int64	Entero con signo de 8 *bytes*
float32	Número de punto flotante de 4 *bytes*
float64	Número de punto flotante de 8 *bytes*
<Método>	Señal a una definición de método o a un miembro referenciado.
<Campo>	Señal a una definición de campo o aun miembro referenciado.
<Tipo>	Señal a una definición de tipo o a untipo referenciado o a un tipo especial.
<Firma>	Señal a una firma independiente.
<String>	Señal a una cadena definida por usuario.

Tabla XV. Tipos de evaluación en pila

Tipo	Descripción
int32	Entero con signo de 4 *bytes*
int64	Entero con signo de 8 *bytes*
Float	Número de punto flotante de 80 *bits*.
&	Puntero manejado o no manejado.
O	Referencia a objetos.
*	Tipo no especificado.

3.2.2 Instrucciones de manipulación de pila

Dup, esta instrucción duplica el valor en el tope de la pila. Si la pila esta vacía el compilador JIT generará error.

Pop, esta instrucción remueve el valor al tope de la pila., este valor es perdido. Si la pila esta vacía el compilador generará error.

ldc.i4 <int32>, carga un entero de 32 *bytes* en la pila.

ldc.i4.s <int8>, carga un entero de 8 *bytes* en la pila.

ldc.i4.m1, carga -1 en la pila.

ldc.i4.0, carga 0 en la pila.

ldc.i4.1, carga 1 en la pila.

ldc.i4.2, carga 2 en la pila, y así sucesivamente dependiendo del último dígito en la instrucción.

ldc.i8 <int64>, carga un entero de 64 *bytes* en la pila.

ldc.r4 <float32>, carga un flotante de 32 *bytes* en la pila.

ldc.r8 <float64>, carga un flotante de 64 *bytes* en la pila.

ldind.i1, carga un entero con signo de 1 *byte* desde una ubicación específica indicada por un puntero desde la pila.

ldind.u1, carga un entero sin signo de 1 *byte*.

ldind.i2, carga un entero con signo de 2 *bytes*.

ldind.u2, carga un entero sin signo de 2 *bytes*.

ldind.i4, carga un entero con signo de 4 *bytes*.
ldind.u4, carga un entero sin signo de 4 *bytes*.
ldind.i8, carga un entero con signo de 8 *bytes*.
ldind.i, carga un entero nativo, un entero de tamaño de un puntero.
ldind.r4, carga un valor de puntero a un flotante de precisión simple.
ldind.r8, carga un valor de puntero a un flotante de precisión doble.
ldind.ref, carga una referencia a objeto.

3.2.3 Operadores aritméticos

add, suma los dos números en la parte superior de la pila.
sub, resta los dos números en la parte superior de la pila.
mul, multiplica los dos números en la parte superior de la pila. El resultado puede tener dos valores especiales en infinito y el NaN, donde NaN es la multiplicación del infinito por cero.
div, esta es la división, y para los enteros la división por cero resulta en una excepción, utilizando también resultados especiales como el NaN, es resultado de 0 dividido 0, así como infinito dividido infinito, así como la división de cualquier cosa por infinito el resultado es cero.
div.un, esta es la división de enteros sin signo.
rem, este es el residuo de una división, donde también se utiliza el NAN, que es resultado en el residuo de infinito y cualquier número, así como entre cualquier número y cero.
rem.un, este es el residuo sin signo, utilizado únicamente para enteros.
neg, esta es la operación de negación de un signo de un número.
add.ovf, es otro tipo de suma, la cual permite sobrecarga.
add.ovf.un, es la sumatoria con sobrecarga para operadores sin signo.
sub.ovf, otro tipo de substracción, la cual permite sobrecarga.
sub.ovf.un, es la substracción con sobrecarga para operadores sin signo.

mul.ovf, otro tipo de multiplicación, la cual permite sobrecarga.
mul.ovf.un, es la multiplicación con sobrecarga para operadores sin signo.

3.2.4 Operadores lógicos

and, esta es la operación lógica *AND* en forma binaria para dos operadores.
or, esta es la operación lógica *OR* en forma binaria para dos operadores.
xor, esta es la operación lógica *OR* exclusivo en forma binaria para dos operadores.
not, esta es la operación lógica de negación en forma binaria para dos operadores.

3.2.5 Corrimientos de valores

shl, corrimiento a la izquierda.
shr, corrimiento a la derecha.
Shr.un, corrimiento a la derecha, tratando el valor como un número sin signo.

3.2.6 Conversión de valores

conv.il, conversión de un valor a un entero de 8 *bytes*.
conv.ul, conversión de un valor a un entero sin signo de 8 *bytes*.
conv.i2, conversión de un valor a un entero de 16 *bytes*.
conv.u2, conversión de un valor a un entero sin signo de 16 *bytes*.
conv.i4, conversión de un valor a un entero de 32 *bytes*.
conv.u4, conversión de un valor a un entero sin signo de 32 *bytes*.
conv.i8, conversión de un valor a un entero de 64 *bytes*.
conv.u8, conversión de un valor a un entero sin signo de 64 *bytes*.
conv.i, conversión de un valor a un entero nativo.
conv.u, conversión de un valor a un entero nativo sin signo.
conv.r4, conversión de un valor a un flotante de 4 *bytes* o *float32*.
conv.r8, conversión de un valor a un flotante de 8 *bytes* o *float68*.
conv.r.un, conversión de un valor entero sin signo a un punto flotante.

Para que las operaciones tengan soporte de sobrecarga, hay que utilizar el ovf como valor intermedio al comando ***conv*** y a su especificación (**i1,u1,i2,u2,etc.**) por ejemplo en ***conv.i1*** pasaría a ser ***conv.ovf.i1***.

3.2.7 Operaciones de chequeo de condiciones lógicas

ceq, chequeo entre dos valores en la pila, para ver si son equivalentes.
cgt, chequeo entre dos valores en la pila, para ver si el primero es mayor que el segundo.
cgt.un, chequeo entre dos valores en la pila, para ver si el primero es mayor que el segundo, asumiendo los operadores como valores sin signo.
ctl, chequeo entre dos valores en la pila, para ver si el primero es menor que el segundo.
ctl.un, chequeo entre dos valores en la pila, para ver si el primero es menor que el segundo, asumiendo los operadores como valores sin signo.

3.2.8 Carga de los argumentos para los métodos

ldarg <unsigned int16>, carga en la pila un argumento entero sin signo.
ldarg.s <unsigned int8>, carga un parámetro corto desde el **ldarg**.
ldarg.0, cargar en la pila el argumento número 0.
ldarg.1, cargar en la pila el argumento número 1.
ldarg.2, cargar en la pila el argumento número 2.
ldarg.3, cargar en la pila el argumento número 3.

3.2.9 Carga de los variables locales

ldloc<unsigned int16>, carga a la pila el valor de una variable local.
ldloc.s <unsigned int8>, carga un parámetro corto desde el **ldloc**.
ldloc.0, cargar en la pila el valor de la variable local número 0.
ldloc.1, cargar en la pila el valor de la variable local número 1.
ldloc.2, cargar en la pila el valor de la variable local número 2.
ldloc.3, cargar en la pila el valor de la variable local número 3.

3.2.10 Llamadas a métodos

jmp <token>, esta instrucción permite el salto del método actual a un método destino especificado a través de su respectivo nombre, en el parámetro respectivo.
call <token>, esta es una llamada a un método no virtual. También se puede llamar a un método virtual pero en este caso solo se puede realizar a través de tablas virtuales de instancias pero con los tipos especificados en esta.
callvirt <token>, esta es una llamada a un método virtual especificado. Este tipo de llamada a métodos es conducida a través de tablas virtuales de instancias.
Ldftn <token>, cargar el puntero a funciones del método especificado por el parámetro, siendo este del tipo MethodDeg o MemberRef.
ldvitrftn <token>, extrae la referencian a un objeto desde la pila, y carga el puntero a la función del método especificado.
calli <token>, extrae un puntero a función del apila, así como todos los argumentos, y hace una llamada directa a un método a través de la firma indicada en el parámetro de esta instrucción.
Tail, marca la siguiente instrucción de llamada como una llamada encolada. Esta instrucción no tiene parámetros y no trabaja con la pila.

3.2.11 Direccionamiento a clases y valores de tipos

ldnull, carga en la pila un objeto de referencia nulo.

ldobj, carga en la pila un valor para un tipo especificado en el parámetro. Esta instrucción extrae de la pila el puntero manejado para la instancia a ser cargada.

stobj, extrae el valor del tipo de la pila.

Ldstr <token>, carga una referencia a una cadena de caracteres, recibiendo como parámetro una cadena definida por el usuario.

cpboj, copia el valor de un tipo de valor a otra instancia. Esta instrucción extrae los punteros a la instancia origen y destino, y no envía nada a la pila.

newobj <token>, coloca en memoria una nueva instancia de una clase, y llama al método constructor de la instancia especificada a través del parámetro de esta instrucción.

initobj <token>, inicializa el tipo de valor de una instancia.

castclass <token>, castea una instancia de clase a una clase especificada en el parámetro.

4. CONSTRUCCIÓN DE APLICACIONES ASP.NET

4.1 Componentes y formularios de ASP.Net

El ambiente de trabajo de la páginas Web de ASP.NET es un modelo de programación escalable basado en el CLR que es la base del lado del servidor para la generación de páginas dinámicas. Es a su vez concebido como una evolución lógica de ASP puro (ASP.NET proporciona compatibilidad sintáctica con las páginas existentes), ya que este está ha diseñado específicamente para tratar varias deficiencias clave del modelo anterior. En forma específica algunas de las características que son proporcionadas por el ASP.Net son:

- Capacidad para crear y utilizar controles de la interfaz de usuario reutilizables que puedan encapsular funcionalidades comunes y, reducir el código que tiene que escribir el programador de una página.
- Capacidad para que los programadores puedan estructurar limpiamente la lógica de la página de forma ordenada (no revuelta).
- Capacidad para que las herramientas de desarrollo proporcionen un fuerte soporte de diseño **WYSIWYG** (lo que ve es lo que se imprime) a las páginas.

Las páginas de formularios *Web* de ASP.NET consisten en archivos de texto con una extensión de nombre de archivo **.aspx**. Pueden implementarse por todo un árbol de directorio raíz virtual IIS. Cuando un cliente solicita recursos .aspx, el motor en tiempo de ejecución de ASP.NET analiza y compila el archivo de destino en una clase de .NET *Framework*, la cual puede es utilizada para procesar de forma dinámica las solicitudes entrantes.

El ASP.Net permite la utilización de los bloques convencionales de los ASP, sin embargo también permite la utilización se controles de servidor, los cuales facilitan mucho la programación de páginas Web. Los controles de servidor se declaran dentro de un .aspx mediante etiquetas personalizadas o etiquetas HTM intrínsecas que contiene un valor de atributo runat="server". Las etiquetas HTML intrínsecas las controla el control de espacio de nombres ***System.Web.UI.HtmlControls***. A cualquier etiqueta que no esté explícitamente asignada a uno de los controles se le asigna el tipo de ***System.Web.UI.HtmlControls.HtmlGenericControl***. Debe tenerse en cuenta que estos controles de servidor mantienen automáticamente cualquier valor introducido por el cliente entre acciones de ida y vuelta al servidor. El estado del control no se almacena en el servidor, sino en un campo del formulario ***<input type="hidden">*** que recibe acciones de ida y vuelta entre solicitudes. Hay que tener en cuenta que no se necesita ninguna secuencia de comando en el cliente. Además de admitir controles estándar de entrada HTML, ASP.NET permite a los programadores utilizar controles personalizados enriquecidos en las páginas, ya que permite la mezcla también con lenguajes de alto nivel tales como Visual Basic, C#, C++, y cualquier otro lenguaje orientado a .Net, lo que permite la utilización de código mucho más complejo que el manejado a través de scripts, todo esto como resultado de que cada control de servidor ASP.NET puede exponer un modelo de objeto con propiedades, métodos y eventos. Los programadores de ASP.NET pueden utilizar este modelo de objeto para modificar e interactuar limpiamente con la página.

Otra característica importante de la programación de formularios Web de ASP.NET es que este proporciona controles de servidor de validación que proporcionan a su vez un modo sencillo y potente de comprobar errores en los formularios de entrada.

4.1.1 Controles ASP.Net y su funcionamiento

Todos los controles de servidor HTML de ASP.NET y los controles de servidores Web se han diseñado para proporcionar compatibilidad de primera clase con los estilos de hojas tipo cascadas (CSS), que ha sido una tendencia muy adoptada en la creación de página *Web*. Debido a esto una de las partes más interesantes del ASP.Net es el saber ver como se utilizan estilos en unión con controles de servidor.

Las etiquetas HTML estándar admiten CSS a través de un atributo de estilo que se puede establecer en una lista de pares atributo-valor delimitada por punto y coma. Todos los controles de servidor HTML de ASP.NET pueden aceptar estilos exactamente del mismo modo que las etiquetas HTML estándar. CSS también define un atributo de clase que se puede establecer en una definición de estilo CSS incluida en una sección <style>...</style> del documento. Los atributos de clase facilitan la definición de estilos y la aplicación de los mismos a varias etiquetas sin tener que volver a definir el mismo estilo. Los estilos de controles de servidor HTML también se pueden establecer de esta forma, lo que permite una gran facilidad en su adaptación, muestra de esto es que cuando se analiza una página ASP.NET, la información de estilo se llena en una propiedad ***Style***, que el de tipo ***CssStyleCollection*** y que pertenece al espacio de nombre ***System.Web.Ul.HtmlControls.HtmlControl***. Básicamente, esta propiedad consiste en un diccionario que expone los estilos del control como colección de valores, indexada por cadenas para cada clave de atributo de estilo.

Un ejemplo de lo ya mencionado antes, es que se puede utilizar código de Visual Bacic para establecer y, en consecuencia, recuperar el atributo de estilo ***width*** en un control de servidor ***HtmlInputText***.

```
<script language="VB" runat="server" >
   Sub Page_Load  (Sender As Object, E As EventArgs)
      TxtData.Style("width") = "90px"
      Response.Write(MyText.Style("width"))
   End Sub
</script>
<input type="text" id="TxtData" runat="server"/>
```

Código VB .Net

Los controles de servidor Web proporcionan un nivel adicional de compatibilidad con estilos mediante la adición de varias propiedades con establecimiento inflexible de tipos para la configuración del estilo habitual, como el color de fondo y de primer plano, el nombre y tamaño de fuente, el ancho, el alto, etc. Estas propiedades de estilo representan un subconjunto de comportamientos de estilo disponible en HTML y se representan como propiedades "planas" expuestas directamente en la clase base ***System.Web.UI.WebControls.WebControl***. La ventaja de utilizar estas propiedades es que proporcionan comprobación de tipo en tiempo de compilación y finalización de instrucciones en herramientas de programación como Microsoft *Visual Studio* .NET.

El espacio de nombres ***System.Web.UI.WebControls*** incluye una clase base ***Style*** que encapsula atributos de estilo comunes (clases de estilo adicionales, como ***TableStyle*** y ***TableItemStyle***, heredadas desde esta clase base común). Numerosos controles de servidor Web exponen propiedades de este tipo para especificar el estilo de elementos de procesamiento individuales del control. Por ejemplo, el control ***WebCalendar*** expone muchas de esas propiedades de estilo: ***DayStyle***, ***WeekendDayStyle***, ***TodayDayStyle***, ***SelectedDayStyle***, ***OtherMonthDayStyle*** y ***NextPrevStyle***. Se pueden establecer propiedades individuales de estos estilos mediante la sintaxis de subpropiedad ***PropertyName-SubPropertyName***.

Como sucede con los controles de servidor HTML, se pueden aplicar estilos a controles de servidor Web mediante una definición de clase CSS. La clase base ***WebControl*** expone una propiedad ***String*** denominada ***CssClass*** para establecer la clase de estilo, lo que permite que si se establece un atributo en un control de servidor que no se corresponde con ninguna propiedad con establecimiento inflexible de tipos, el atributo y el valor se llenan en la colección ***Attributes*** del control. De forma predeterminada, los controles de servidor procesarán estos atributos no modificados en el HTML devuelto al cliente del explorador solicitante. Esto significa que los atributos de estilo y de clase se pueden establecer directamente en controles de servidor Web en vez de utilizar las propiedades con establecimiento inflexible de tipo. Mientras que esto requiere entender algo del procesamiento real del control, también puede constituir una forma flexible de aplicar estilos, lo cual puede ser muy útil con los controles estándar de entrada de formulario. Los estilos de controles de servidor Web también se pueden establecer mediante programación con el método ***ApplyStyle*** de la clase base ***WebControl***, definiéndolo en forma más ordenada y basada en eventos más definidos.

```
<script language="VB" runat="server">
   Sub Page_Load  (Src As Object, E As EventArgs)
```

```
        Dim Estilo As New Style
        Estilo.BorderColor = Color.Black
        Estilo.BorderStyle = BorderStyle.Dashed
        Estilo.BorderWidth = New Unit(1)
        IDUsuario.ApplyStyle (MyStyle)
        Clave.ApplyStyle (MyStyle)
        CmdIngreso.ApplyStyle (MyStyle)
    End Sub
</script>
Login:
<ASP:TextBox id="IDUsuario" runat="server" />/<p/>
Password:
<ASP:TextBox id="Clave" TextMode="Password" runat="server" />
View:
<ASP:DropDownList id="CmdIngreso" runat="server">
 ...
</ASP:DropDownList>
```

Código VB .Net

4.1.2 Desarrollo de formularios *web*

El elemento más importante en el desarrollo de formularios, es obviamente el manejo de la sintaxis de formularios *web* de ASP.NET, y relacionado a esto podemos concretizar claramente que una página de formularios *web* de ASP.NET consiste en un archivo de texto declarativo con una extensión de nombre de archivo .aspx. Además del contenido estático, de los cuales se pueden utilizar ocho elementos distintos de marcado de sintaxis.

Procesar sintaxis de código: <% %> y <%= %> , consiste en bloques de procesamiento de código se indican con elementos <% ... %> (permiten al usuario controlar de forma personalizada la emisión de contenido) y se ejecutan durante la fase de procesamiento de la ejecución de páginas de formularios Web. En el siguiente ejemplo se muestra cómo utilizarlos para ascender en bucle por contenido HTML.

```
<% For I=0 To 7 %>
    <font size="<%=i%>"> Hola Mundo </font> <br>
<% Next %>
```

El código que se encuentra entre <% ... %> sólo se ejecuta, mientras que las expresiones que influyen un signo de igual <%= ... %>, se evalúan y el resultado se emite como contenido.

Código de declaración: <*script runat="server"*>, consisten en bloques de declaración de código definen métodos y variables miembro que se compilarán en la clase ***Page*** generada. Estos bloques pueden utilizarse para crear lógica de página o desplazamiento. Esta funcionalidad es precisamente la que se ha expuesto en la información relacionada con controles.

Control de servidor ASP.Net, son los controles personalizados que permiten a los programadores de páginas generar dinámicamente interfaces de usuario HTML y responder a solicitudes de clientes. Se representan en un archivo mediante una sintaxis declarativa basada en etiquetas. Estas etiquetas se distinguen de otras etiquetas porque contienen un atributo ***"runat=server"***. En el siguiente ejemplo se muestra cómo se puede utilizar un control de servidor ***<asp:label runat="server">*** en una página ASP.NET. Este control se corresponde con la clase ***Label*** del espacio de nombres ***System.Web.Ul.WebControls*** que se incluye de forma predeterminada.

Control de servidor HTML ASP.Net, son controles que permiten a los programadores de páginas manipular mediante programación elementos HTML dentro de una página. Una etiqueta de control de servidor HTML se distingue de los elementos HTML del cliente mediante un atributo ***"runat=server"***. En el siguiente ejemplo se muestra cómo se puede utilizar un control de servidor **<*span runat=server*>** HTML en una página ASP.NET. De la misma forma que sucede con otros controles de servidor, se puede tener acceso mediante programación a sus métodos y propiedades

Enlace de datos: <%# %> , la compatibilidad de enlace de datos integrada en ASP.NET permite a los programadores de páginas enlazar jerárquicamente las propiedades del control con los valores del contenedor de datos. El código ubicado en un bloque de código <%# %> sólo se ejecuta cuando se invoca al método ***DataBind*** del contenedor del control primario correspondiente. Al llamar al método ***DataBind*** de un control, se provoca que un árbol recursivo vaya desde dicho control hacia la parte de abajo del árbol, se provoca el evento ***DataBinding*** en cada control de servidor de esa jerarquía y se evalúan las expresiones de enlace de datos del control en consecuencia. Así pues, si se llama al método ***DataBind*** de la página, entonces se llamará a todas las expresiones de enlace de datos de la página.

Etiquetas de objetos: <*object runat="server"* />, estas permiten a los programadores de páginas declarar y crear instancias de variables mediante una sintaxis declarativa basada en etiquetas.

Comentarios en el cliente: <%-- Comentario --%>, permiten a los programadores de páginas evitar que ejecute o se procese el código del servidor (controles de servidor incluidos) y el contenido estático.

Inclusiones en el cliente: <-- *#Include File="Locaton.inc"* -->, estas marcas en el servidor permiten a los programadores insertar contenido sin formato de un archivo especificado en cualquier parte de la página ASP.NET.

Además del procesamiento de los diversos comandos incluidos en estas areas de sintaxis, también se puede hacer la separación de código, utilizando en el archivo **.aspx** la información orientada a HTML, y utilizar los eventos para los controles utilizados en un archivo cuya extensión seria **.vb** en el caso de visual basic, donde se incluyen todos los comandos y procesos que son enmarcados dentro de los scripts como ocurre en el tipo de declaración ***<script runat="server">***, permitiendo con esto mayor orden y entendimiento de códigos.

Debido a la validez de las bibliotecas del .Net *Framework* en las páginas ASP.Net, es preciso indicar que la programación de estas páginas es muy similar a la programación de programas diseñados para Windows, pero con flexibilidad de ser ejecutado desde cualquier browser, bajo la única condicionante de que el servidor de *Web* debe tener instalado el .Net Framework, para que este se encargue de realizar la compilación JIT necesaria al momento de recibir solicitudes de la página, y ejecutado la página que cabe decir no es igual a una ASP convencional en cuanto a ejecución, pues en este tipo se realiza una compilación que permite mayor versatilidad en los controles utilizados, así como el elaborar cosas más complejas en forma breve, y continuar permitiendo la apertura de plataformas del lado del cliente.

Otro elemento que permite gran versatilidad en ASP.Net son las validaciones de los formularios, ya que el marco de desarrollo de formularios *Web* incluye un conjunto de controles de servidor de validación que proporcionan un modo sencillo a la vez que potente de comprobar errores en los formularios de entrada y, en caso necesario, mostrar mensajes al usuario. Estos controles de validación se agregan a una página de formularios *Web* con otros controles de servidor. Existen controles para tipos concretos de validación, como la comprobación de intervalos o la coincidencia de modelos, además de ***RequiredFieldValidator***, que se asegura de que un usuario omita un campo de entrada, los cuales pueden ejecutarse sin necesidad de viajar toda la información al servidor y posteriormente retornar los errores en el cliente, lo cual ayuda en la optimización de las aplicaciones.

Los controles de validación funcionan con un subconjunto limitado de controles de servidor HTML y *Web*. Para cada control, una propiedad específica contiene el valor que se va a validar.

Tabla XVI. Propiedades de controles de valicación formularios *web*

Control	Propiedad de validación
htmlinputtext	*value*
htmltextarea	*value*
htmlselect	*value*
htmlinputfile	*value*
textbox	*text*
listbox	*selecteditem.value*
dropdownlist	*selecteditem.value*
radiobuttonlist	*selecteditem.value*

El formulario de validación más sencillo es un campo requerido. Si el usuario introduce cualquier valor en un campo, es válido. Si todos los campos de la página son válidos, la página es válida, para realizar esto se utiliza el control ***RequiredFieldValidator***, pero de igual forma que este también hay controles de validación para tipos de validación específicos, como la comprobación de intervalos o la coincidencia de modelos.

Tabla XVII. Controles de validación formulatios *web*

Nombre del control	Descripción
requiredfieldvalidator	Se asegura de que el usuario no omita entradas.
comparevalidator	Compara una entrada de usuario con un valor constante o un valor de propiedad de otro control mediante un operador de comparación.
rangevalidator	Comprueba que una entrada de usuario se encuentra entre los límites superior e inferior especificados.
regularexpressionvalidator	Comprueba que la entrada coincide con un patrón definido por una expresión regular. Permite comprobar secuencias de caracteres previsibles, bajo un formato establecido
customvalidator	Comprueba la entrada del usuario mediante lógica de validación que codifica el usuario. Permite comprobar los valores derivados en tiempo de ejecución.
validationsummary	Muestra los errores de validación en forma de resumen para todos los validadores de la página.

Los controles de validación siempre realizan una comprobación de validación en el código del servidor. No obstante, si el usuario trabaja con un explorador que admite DHTML, los controles de validación también pueden realizar validaciones mediante secuencias de comandos del cliente. Con validación en el cliente, se detectan algunos errores en el cliente cuando se envía el formulario al servidor. Si resulta que alguno de los validadores es un error, se cancela el envío del formulario al servidor y se muestra la propiedad ***Text*** del validador. Esto permite al usuario corregir lo escrito antes de enviar el formulario al servidor. Los valores de campo se revalidan en cuanto el campo que contiene el error pierde el foco, proporcionando así al usuario una rica experiencia de validación interactiva. Es importante mencionar que para los formularios Web siempre realiza la validación en el servidor, incluso cuando ya se ha realizado en el cliente, para evitar que los usuarios puedan omitir la validación mediante la suplantación de otro usuario o de una transacción previamente aprobada. La validación en el cliente se ha habilitado de forma predeterminada. Si el cliente tiene la capacidad, la validación de alto nivel se realiza automáticamente. Para deshabilitar la validación en el cliente, es necesario establecer la propiedad ***ClientTarget*** de la página en ***Downlevel***, y para habilitarla en ***Uplevel***.

Cuando se procesa la entrada del usuario (por ejemplo, cuando se envía el formulario), el marco de trabajo de la página de formularios *Web* pasa la entrada del usuario al control o controles de validación asociados. Los controles de validación comprueban la entrada del usuario y establecen una propiedad para indicar si la entrada pasó la prueba de validación. Tras haber procesado todos los controles de validación, se establece la propiedad ***IsValid*** de la página; si cualquiera de los controles muestra que se produjo un error en la validación, toda la página se establecerá como inválida.

Si un control de validación da error, un mensaje de error aparecerá en la página por ese control de validación o en un ***ValidationSummary*** en cualquier otra parte de la página. Se visualiza el control ***ValidationSummary*** cuando la propiedad ***IsValid*** de la página es falso. Sondea cada control de validación de la página y agrega mensajes de texto expuestos por cada uno. Otro control de servidor para validaciones es el ***CompareValidator*****,** el cual compara los valores de dos controles, este utiliza tres propiedades clave para realizar la validación. ***ControlToValidate*** y ***ControlToCompare*** contienen los valores que se van a comparar. ***Operator*** define el tipo de comparación que se va a realizar, como lo puede ser Equal o Not Equal.

El control de servidor ***RangeValidator*** prueba si un valor de entrada produce un error en un intervalo dado. Este utiliza tres propiedades clave para realizar la validación. ***ControlToValidate*** contiene el valor que hay que validar. ***MinimumValue*** y ***MaximumValue*** definen los valores máximo y mínimo del intervalo válido.

El control de servidor ***RegularExpressionValidator*** confirma que la entrada coincide con un patrón definido por una expresión regular. Este tipo de validación permite comprobar secuencias de caracteres previsibles, como las de los números de la seguridad social, las direcciones de correo electrónico, los números de teléfono y los códigos postales, entre otras. Este utiliza dos propiedades clave para realizar la validación. ***ControlToValidate*** contiene el valor que hay que validar. ***ValidationExpression*** contiene la expresión regular con la que debe coincidir.

El control de servidor ***CustomValidator*** llama a una función definida por el usuario para realizar validaciones que los validadores estándar no pueden controlar. La función personalizada puede ejecutarse en el servidor o en la secuencia de comandos del cliente, como JScript o VBScript. Para una validación personalizada en el cliente, el nombre de la función personalizada debe identificarse en la propiedad ***ClientValidationFunction***. La función personalizada debe presentar una estructura como:

function Validacion (Origen, Argumentos).

Debe observarse que **Origen** es el objeto ***CustomValidator*** del cliente, mientras que **Argumentos** es un objeto con dos propiedades, ***Value*** e ***IsValid***. La propiedad ***Value*** es el valor que se validará y la propiedad ***IsValid*** es el lógico (falso / verdadero) utilizado para establecer el resultado devuelto de la validación.

4.2 Manejo de datos con ASP.Net

El acceso a datos es la esencia de cualquier aplicación del mundo real, por lo cual ASP.Net proporciona una buena cantidad de controles que se integran bien con las API de acceso administrado a datos que proporciona CLR. Uno de estos controles, y posiblemente el más frecuentemente utilizado es el ***DataGrid*** de ASP.NET para enlazar con los resultados de consultas SQL y archivos de datos XML. El acceso a datos en el servidor es exclusivo en cuanto a la ausencia básica de información de estado de las páginas *Web*, lo que presenta algunas dificultades al intentar realizar transacciones como insertar o actualizar registros a partir de un conjunto de datos recuperados desde una base de datos, sin embargo controles como los ***DataGrid*** pueden ayudar a administrar estas dificultades, lo que permite concentrarse más en la lógica de la aplicación y menos en los detalles de la administración del estado y el control de eventos.

El CLR proporciona un conjunto completo de interfaces API de acceso a datos administrados para el desarrollo de aplicaciones con un uso intensivo de datos. Estas API ayudan a extraer datos y presentarlos de forma coherente sin importar su origen (SQL *Server*, OLEDB, XML, entre otros). Existen tres objetos con los que más se trabaja: conexiones, comandos y conjuntos de datos.

- Una conexión que es la que representa una conexión física a algún almacén de datos, como cualquier base de datos o un archivo XML.
- Un comando que es el que representa una directiva para recuperar (*select*) desde el almacén de datos o manipular el almacén de datos (*insert, update, delete*).
- Un conjunto de datos representa los datos reales con los que trabaja una aplicación.

Se debe de tener en cuenta que los conjuntos de datos se encuentran siempre desconectados de la conexión de origen y el modelo de datos correspondiente, y que pueden modificarse de forma independiente. Sin embargo, los cambios que se realizan en un conjunto de datos pueden reconciliarse fácilmente con el modelo de datos originario.

La gran ventaja de utilizar un conjunto de datos consiste en que proporciona una vista desconectada de la base de datos. Se puede operar con un conjunto de datos de la aplicación y, posteriormente, cotejar los cambios con la base de datos real. En el caso de aplicaciones cuya ejecución es de larga duración, éste suele ser el mejor enfoque. En el caso de aplicaciones Web, se suelen realizar operaciones breves con cada solicitud (normalmente, sólo para visualizar los datos).

4.2.1 ADO.Net y su base en el XML

ADO.Net es una evolución del modelo de acceso a datos de ADO que controla directamente los requisitos del usuario para programar aplicaciones escalables. Se diseñó teniendo en cuenta la escalabilidad, la independencia y el estándar XML, y realmente es una evolución muy fuerte que incorpora nuevos conceptos y nuevas formas de utilización de objetos de control de datos, y más si consideramos que esta basado en XML, esto lo hace muy flexible y principalmente estratégico para el Internet. Algunas ideas evolucionaron en ADO.Net, tales como el que este suministre un objeto DataReader, el cual viene a ser el espejo de la función *OpenSQLForwardOnly*, de lenguajes como Visual Basic, de igual forma equivalente o pariente más cercano del *Recordset* es el DataSet, el cual ya incorpora muchas características más poderosos en el objeto ya que no solo maneja conjuntos de registros sino también relaciones entre estos, pudiendo manejar una gran cantidad de tabla en una sola estructura, lo cual permite una gran facilidad de navegación de datos.

ADO.Net utiliza algunos objetos ADO, como ***Connection*** y ***Command***, y también agrega objetos nuevos. Algunos de los nuevos objetos clave de ADO.Net son ***DataSet***, ***DataReader*** y ***DataAdapter***. Y la diferencia más importante entre esta fase evolucionada y las arquitecturas de datos anteriores es que existe un objeto, ***DataSet***, que es independiente y diferente de los almacenes de datos. Por ello, ***DataSet*** funciona como una entidad independiente. Se puede considerar un *DataSet* como un conjunto de registros que desconectado y que no sabe nada sobre el origen y el destino de los datos que contiene.

El objeto ***DataAdapter*** es el objeto que se conecta a la base de datos para llenar el objeto ***DataSet***. A continuación, se vuelve a conectar a la base de datos para actualizar los datos de dicha base de datos a partir de las operaciones realizadas en los datos contenidos en el objeto ***DataSet***. En el pasado, el procesamiento de datos se basaba principalmente en la conexión. Ahora, con el fin de proporcionar a las aplicaciones multinivel mayor eficacia, se está adoptando para el procesamiento de datos un enfoque basado en mensajes que manipulan fragmentos de información. En el centro de este enfoque se sitúa el objeto ***DataAdapter***, que proporciona un puente entre un objeto ***DataSet*** y un almacén de datos para recuperar y guardar datos, para ello, envía solicitudes a los comandos SQL apropiados que se ejecutan en el almacén de datos.

El objeto ***DataSet*** basado en XML proporciona un modelo de programación coherente que funciona con todos los modelos de almacenamiento de datos: sin formato, relacional o jerárquico. Independientemente del origen de los datos del objeto ***DataSet***, éstos se manipulan mediante el mismo conjunto de API estándar expuestas a través del objeto ***DataSet*** y sus objetos subordinados. Aunque el objeto ***DataSet*** no tiene conocimiento del origen de sus datos, el proveedor administrado tiene información detallada y específica, debido a que controla por si mismo los metadatos de la estructura. La función del proveedor administrado es conectar, llenar y almacenar el objeto ***DataSet*** desde almacenes de datos o viceversa.

Los proveedores de datos . OLEDB y *SQLServer* (*System.Data.OleDb y System.Data.SqlClient*) forman parte del *Framework* como objetos ***Conecction*** y son utilizados específicamente establecer la comunicación con bases de datos, estas se representan mediante clases específicas de proveedor, como ***OleDBConnection*** o ***SQLConnection***, cuya parametrización se hace a través de una cadena de conexión.

Los objetos ***Command*** contienen la información que se envía a una base de datos y se representan mediante clases específicas de un proveedor, como ***SQLCommand*** u ***OleDBCommand***, los cuales podrían ser una llamada a un procedimiento almacenado, una instrucción *UPDATE* o una instrucción que devuelve resultados. También es posible utilizar parámetros de entrada o de resultados y devolver valores como parte de la sintaxis del comando.

Los objetos ***DataReader*** permite lectura de registros solamente hacia delante y el ***DataAdapter*** que permite la relación entre la fuente de datos y los objetos de manipulación desconectada, estos objetos en cierto modo, son un sinónimo de un cursor de sólo lectura y sólo hacia delante para datos. La API de ***DataReader*** es compatible con datos sin formato y con datos jerárquicos. Cuando se ejecuta un comando en la base de datos, se devuelve un objeto ***DataReader***. El formato del objeto ***DataReader*** devuelto es distinto de un conjunto de registros. Por ejemplo, podría utilizarse el objeto ***DataReader*** para mostrar los resultados de una lista de búsqueda en una página *Web*.

Los objetos ***DataSet*** (posiblemente el más importante), son los que permiten almacenar datos en forma de XML, así como datos relacionales, así como para configurar el acceso remoto y programar sobre datos de este tipo. Como ya se mencionó es similar al objeto ***Recordset*** de ADO, pero más eficaz y con una diferencia importante: ***DataSet*** siempre está desconectado. El objeto ***DataSet*** representa a una memoria caché de datos, con estructuras análogas a las de una base de datos, como tablas, columnas, relaciones y restricciones.

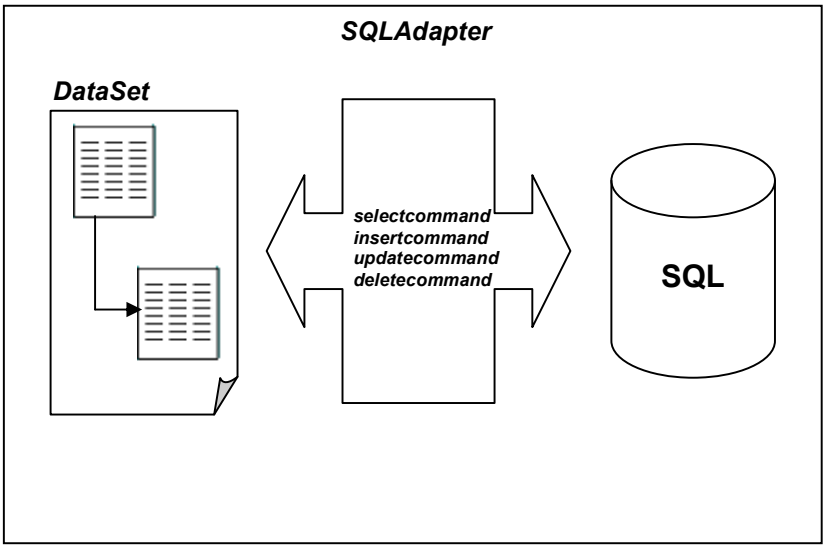
Aunque se puede utilizar un objeto ***DataSet*** como una base de datos (y su comportamiento es muy similar), es importante recordar que los objetos ***DataSet*** no interactúan directamente con bases de datos ni con otros datos de origen. Esto permite al programador trabajar con un modelo de programación que siempre es coherente, independientemente de dónde resida el origen de datos. En los objetos ***DataSet*** se pueden colocar datos provenientes de una base de datos, un archivo XML, código o información escrita por el usuario. A continuación, a medida que se realizan cambios en el objeto ***DataSet***, se puede hacer un seguimiento y una comprobación de los cambios antes de actualizar los datos de origen. El método ***GetChanges*** del objeto ***DataSet*** crea en realidad otro objeto ***DataSet*** que sólo contiene los cambios realizados en los datos. Posteriormente, un objeto ***DataAdapter*** u otros objetos, utilizan este objeto ***DataSet*** para actualizar el origen de datos original.

El objeto ***DataSet*** tiene muchas características de XML, incluida la capacidad de producir y consumir datos XML y esquemas XML. Los esquemas XML se pueden utilizar para describir esquemas intercambiables a través de servicios *Web*. De hecho, un objeto ***DataSet*** con un esquema puede compilarse con seguridad de tipos y finalización automática de instrucciones.

Los objeto ***DataAdapter*** funciona como un puente entre el objeto ***DataSet*** y los datos de origen. El uso del objeto ***SqlDataAdapter*** u ***OleDBAdapter*** según el proveedor específico de datos (junto con los objetos ***Command*** y ***Connection*** asociados) permite aumentar el rendimiento global al trabajar con bases de datos.

El objeto ***DataAdapter*** utiliza comandos para actualizar el origen de datos después de hacer modificaciones en el objeto ***DataSet***. Si se utiliza el método ***Fill*** del objeto ***DataAdapter***, se llama al comando *SELECT*; si se utiliza el método ***Update*** se llama al comando *INSERT*, *UPDATE* o *DELETE* para cada fila modificada. Es posible establecer explícitamente estos comandos con el fin de controlar las instrucciones que se utilizan en tiempo de ejecución para resolver cambios, incluido el uso de procedimientos almacenados.

Figura 21. *DataAdapters* y *DataSets*



Los objetos ***DataReader*** proporcionan un puntero de tipo sólo hacia delante de sólo lectura sobre los datos recuperados desde una base de datos. Para utilizarlo se declara un objeto ***Command*** del cual se ejecuta el comando ***ExecuteReader*** que devuelve ***DataReader***.

Al ejecutar comandos sin devolución de datos, se puede utilizar el método ***ExecuteNonQuery*** de los objetos ***Command***, el cual devuelve el número de filas afectadas, asimismo si se quiere leer solamente un dato se puede ejecutar la función ***ExecuteScalar***, la cual devuelve el primer dato de la primera fila que de cómo resultado la consulta.

Es importante mencionar que siempre la conexión a la base de datos no está relacionada directamente con el objeto ***DataAdapter***, sino directamente con cada uno de los objetos ***Command***, por lo cual se pueden tener distintas conexiones hacia un mismo adaptador.

Todos los objetos que contiene datos como el ***DataGrid*****,** los cuales pueden ser leídos de un origen de datos como el ***DataSet*** u otros derivados como el ***Dataview*** admiten una propiedad llamada ***DataSource*** que toma cualquiera de estos objetos de almacenamiento de datos. Asimismo se puede utilizar ***DataSet*** asignando la propiedad ***DefaultView*** de una tabla incluida en ***DataSet*** al nombre de la tabla que se desea utilizar dentro de ***DataSet***. La propiedad ***DefaultView*** representa el estado actual de una tabla dentro de ***DataSet***, incluido cualquier cambio que haya realizado el código de la aplicación (como eliminar filas o cambiar valores). En el caso específico de las aplicaciones para Internet, se necesita no solamente asignar el origen de datos para lo objetos de despliegue, sino también es necesario ejecutar la carga de los mismos, para lo cual se utiliza la llamada al método ***DataBind*****()**.

Como ya se ha mencionado que ***DataSet*** estaba diseñado para datos abstractos de tal forma que es independiente del origen de datos real. Estos objetos admite el método ***ReadXml*** que toma un objeto ***FileStream*** como parámetro. El archivo que se lee en este caso debe contener un esquema y los datos que se desean leer.

4.3 Desarrollo de servicios *web*

En la actualidad los sitios *web* programables adquieren un valor adicional al de aquellos sitios a los que se obtiene acceso de forma pasiva, convirtiéndose en servicios Web reutilizables y programables. El CLR proporciona soporte integrado para crear y exponer servicios *Web*, utilizando una abstracción de programación coherente con programadores de *Web Forms* ASP.NET. El modelo resultante es escalable y ampliable, asimismo comprende estándares abiertos de Internet (HTTP, XML, SOAP) de forma que cualquier cliente con servicios de Internet puede obtener acceso al modelo y lo puede consumir.

Un archivo .asmx (Servicio .Net) es un archivo de texto similar a un archivo .aspx (Asp .Net), y pueden formar parte de una aplicación que incluya ambos tipos, de esta forma, se puede asignar una dirección URL a los archivos .asmx, como se hace con los archivos .aspx. Los servicios Web pueden ser tan simples como devolver un dato, como poder devolver una lista de datos a través de objetos como los Dataset, así como internamente ejecutar funciones muy complejas de manipulación de datos, o tan simples como una suma.

```
<%@ WebService Language="VB" Class="Operaciones" %>
Imports System
Imports System.Web.Services
Public Class Operaciones:
Inherits WebService
    <WebMethod()> Public Function
    Suma (ByVal V1 As Integer, ByVal V2 As Integer) as Integer
        Return(V1+V2)
    End Function
End Class
```

Función pública de tipo método *web*

Como se puede ver el programa inicia con la directiva ***WebService*** ASP.NET y establece el lenguaje en Visual Basic, posteriormente se importa el espacio de nombres ***System.Web.Services***, luego está la declaración de la clase **operaciones**. Esta clase se deriva de la clase base ***WebService***; finalmente se declaran los procedimientos y funciones que sean requeridas, pero los que serán publicados llevan la palabra clave ***<WebMethod()>,*** delante de su firma (firma convencional de métodos .Net).

4.3.1 Estándares de creación de servicios y su comportamiento

Actualmente muchos de los programas para visualización de páginas o interpretes de HTML, ya tiene la habilidad de utilizar el SOAP, tal como pasa en los Microsoft Internet Explorer versión 5 y posteriores. El nuevo comportamiento ***WebService*** permite a la secuencia de comandos en el cliente invocar métodos remotos expuestos por servicios *Web* .Net XML de Microsoft u otros servidores Web que admiten el protocolo SOAP. El comportamiento ***WebService*** se implementa con un archivo HTML Componentes (HTC) como un comportamiento asociado, por lo que puede utilizarse en Internet Explorer. La finalidad de utilizar este tipo de comunicación de datos es poder realizar una estandarización no dependiente, ya que el protocolo SOAP es totalmente abierto y se basa en el XML, lo cual permite que aunque los procesos sean ejecutados en los servidores toda la información de resultados sea enviada en este lenguaje permitiendo que lo pueda visualizar cualquier cliente, sin depender de una aplicación o plataforma específica, por lo cual se puede decir que no existen estándares específicos de desarrollo de las aplicaciones, ya que esto puede ser totalmente libre ya que su ejecución se realiza en el lado del servidor, pero si existe una estandarización de la comunicación, para solicitud y envío de resultados, la cual está estrictamente restringida por un lenguaje que permite gran apertura en la forma de diversificar los datos como lo es el XML.

La finalidad del comportamiento ***WebService*** es proporcionar una manera sencilla de utilizar y aplicar SOAP, sin requerir un conocimiento profundo de su implementación. El comportamiento ***WebService*** admite el uso de una amplia variedad de tipos de datos, incluidos tipos de datos SOAP intrínsecos, matrices y datos XML, que finalmente permite a recuperar información de servicios Web de XML y actualizar una página de forma dinámica mediante el uso de DHTML y secuencias de comandos, sin requerir el desplazamiento ni una actualización de página completa.

La publicación de este tipo de servicios es muy simple, pues consiste únicamente en llevar a un directorio virtual los archivos relacionados con la funcionalidad de los servicios y con eso bastará para que inicien a ser consumidos. Los servicios .Net por default si son llamados como una página Web, mostrarán una página estandarizada que muestra los diversos métodos del servicio y al elegir uno aparecerán las solicitudes de los parámetros necesarios y al ejecutarla se podrá visualizar el resultado de la función, el cual está basado en XML.

4.3.2 Consumo de un servicio *web*

Además de la tecnología de servidor ASP.Net que permite a los programadores crear servicios *web*, .NET *Framework* proporciona conjuntos de herramientas y código sofisticados para consumir servicios *web*. Como los servicios *web* se basan en protocolos abiertos principalmente el SOAP, esta tecnología para cliente también se puede utilizar para consumir servicios *Web* que no estén basados en ASP.Net.

Si se publicara un servicio con un método de suma como el que fue escrito anteriormente, al llamar a este método se empaqueta la solicitud en SOAP a través de HTTP y recibe la respuesta codificada de la misma forma, que será interpretado posteriormente.

Desde la perspectiva del cliente, el código es sencillo, principalmente para los lenguajes orientados a .Net, ya que en este caso el único requisito es direccionar en la aplicación la dirección URL del servicio y adjuntarlo como una clase del proyecto que se está trabajando, donde esta ya será utilizada como un función de una clase local, con la diferencia de que internamente está obteniendo datos en forma remota, un ejemplo de ello sería consumir la función de suma ya mencionada.

Dim Oper as New Operaciones
Dim Resultado As Integer = Operaciones.Suma (15,35)

El valor que tomaría la variable **resultado** sería **50**, de igual forma que si se estuviera utilizando una clase local, solo que ejecutándose el procedimiento en un servidor.

Como se puede ver realmente es fácil consumir los servicios *web*, pues a nivel de programación es bastante transparente, pero algo importante de esto es que realmente se pueden llegar a utilizar servicios tan simples como el mencionado o servicios tan complejos con operatorias en el servidor y que retornen valores simple o tablas de datos bajo un esquema XML, asimismo esta funcionalidad permite que los servicios no estén limitados únicamente a páginas *web*, sino también pueden ser consumidos por aplicaciones para *Windows* basadas en .Net, sin tener que rescribir su código de funcionalidad.

5. DESARROLLO APLICATIVO-COMPARATIVO ASP.NET E ILASSEMBLER

5.1 Planteamiento de aplicación a desarrollar

La aplicación que se presentará será un sitio *web* el cual permitirá a los visitantes utilizar algunas funcionalidades relacionadas con las generalidades del control académico de los estudiantes, para la cual existirán dos panoramas, el primero para el cual no será necesario realizar ninguna autentificación de usuarios, y la segunda para la cual si será necesario, siendo sus funcionalidades las siguientes:

Sin autenticación

- Visualización de los diversos pensums de la facultad de ingeniería.

Con autenticación

- Visualización de los diversos pensums de la facultad de ingeniería.
- Asignación de cursos para el semestre en curso.
- Visualización de boleta de asignación del estudiante.
- Visualización de listado de cursos aprobados del estudiante.

La autentificación de los usuarios, se realizará a través del número de carné del estudiante y un *password* relacionado.

La aplicación es una aplicación corta, sin embargo la estructuración que se utilizará, permitirá hacer una comparación que permita ver por áreas las diferencias en un desarrollo orientado a .Net (páginas ASP.Net) y uno que podriamos llamar tradicional (páginas ASP).

La primera gran diferencia que se tendrá será la creación de un sistema basado en cuatro capas, que a diferencia de un sistema tradicional puro, se manejaria solamente como tres capas, siendo estas:

- Capa de base de datos
- Capa de lógica de funcionalidad
- Capa de servicios de datos
- Capa de presentación

La capa de servicios de datos, generalmente no es utilizada al momento de desarrollar ASPs tradicionales, de utilizarse se caeria a una mezcla que haria al esquema un hibrido. Debido a la comparación que se desea exponer, es importante destacar que por generalidad cuando se desarrollan ASPs, estas llegan a ejecutar funciones en forma directa de una o varias librerias registradas como componentes.

La idea de esta aplicación es básicamente mostrar la funcionalidad de le tecnología .Net a través de diversas capas, y de su interacción, ya que esto nos permitirá ver diferencias claras en el manejo de acciones a base de datos a través de ADO.Net, las cuales se vuelven fundamentales para el manejo de la aplicación y que marcan una gran diferencia respecto al manejo a través de ADO tradicional.

Otra parte imporante de la funcionalidad, es el contenido de la capa de lógica de funcionalidad, que se convertirá en el núcleo de las operaciones del sitio *web*, lo cual marcará una direncia importante en cuanto su manejo interno de comunicación hacia las capas de base de datos y hacia la de servicio de datos.

Un tercer aspecto a mostrar y debatir será el manejo y publicación de datos a través de servicios, así como el manejo serializado de la información publicada, es decir el manejo de la misma a través de XML y el protocolo SOAP.

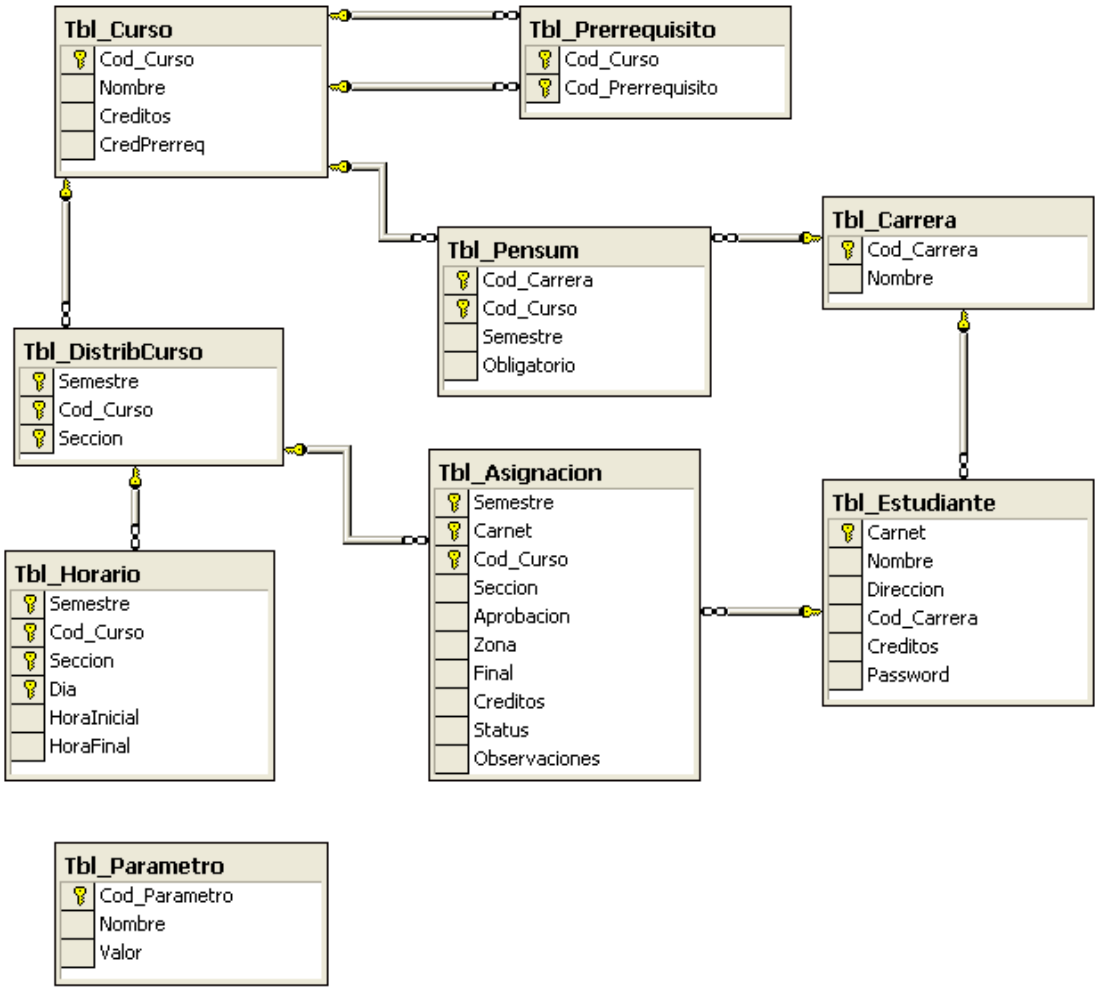
Finalmente se mostrarán algunas de las características del desarrollo a través de formularios activos orientados a .Net, para ver su estructuración y entender su funcionamiento.

5.2 Desarrollo de aplicación

5.2.1 Capa de base de datos

La capa de base de datos es muy importante en la funcionalidad de la aplicación, sin embargo para los objetivos de la demostración no es relavante debido a que esta capa aunque es importante para el almacenamiento de los datos y el manejo de la estructura de la información, su generación y estructura no tiene absolutamente ninguna diferencia cuando que crea una aplicación tradicional y una orientada a .Net, ya que la diferencia fundamental se verá en la cualquier capa que accese a los datos, pero no en la estructura de esta, por lo mismo, para esta capa solamente se presentará la estructura de la base de datos a través de un diagrama entidad relación de la misma.

Figura 22. Diagrama de base de datos de aplicación de control académico



5.2.2 Capa de lógica de funcionalidad

La capa de lógica de funcionalida, posiblemente es la capa que lleva la mayor parte del peso de la aplicación, ya que ella es donde se declaran las diversas clases y funciones que permiten al sistema tomar datos de la base de datos para exponerlos o publicarlos, así como será quien se encargue de manejar transaccionalmente los datos hacia la base de datos.

Existen dos diferencias fundamentales en el manejo de las aplicaciones tradicionales y las aplicaciones orientadas a .Net, siendo una de ellas en cuanto al desarrollo y otra en cuanto a su funcionalidad. La difererencia en tiempo de desarrollo surge, debido a que el poder realizar código orientado totalmente a objetos en las clases, permite que algunas de las características de código sean heredadas desde una clase padre que permite centralizar el manejo de funciones comunes. En cuanto a la diferencia de funcionalidad de las clases para una aplicación tradicional y una de .Net, se puede mencionar caracteristicas importantes en cuanto a su constitución, siendo la generadora de todo el hecho de que el código de las clases, aunque es generado a una librería DLL, no está compilado a lenguaje de máquina, sino simplemente está generado a lenguaje lenguaje intermedio de .Net, lo cual hace que sea ejecutado a través de la compilación del JIT en tiempo de corrida, esto es lo que permite que el manejo de clase sea más libre, ya que al ser un componente de .Net no obliga al sistema operativo a registrar la librería, ya que esta es interpretada por el *Framework*, permitiendo así algunas cosas importantes:

a) Al ser un componente, contenedor de funciones y no ser registrado, permite la convivencia de diversas versiones y aplicaciones.

b) El componente puede ser accesado por aplicaciones de tipo *Windows* o de tipo *Web*, teniendo la misma funcionalidad.

c) La funcionalidad a través del *Framework*, permite que el manejo de la clase se haga unicamente en el equipo en el que se encuentra el componente, sin importar el origen de la solicitud de información, siendo esto un punto fuerte de la construcción de este tipo de clases, ya que dicha transparencia permite que los componentes funciones en forma transaccional hacia la base de datos. Esta necesidad transaccional obliga en componentes que no son orientados a .Net a que los componentes sean registrados y también sean administrados a través del Microsoft *Transaction Server*, lo cual en una aplicación orientada a .Net desaparece, lo cual hace mayor la viabilidad de que las clases sean reutilizadas tanto para aplicaciones *Windows* como para *Web*, ya que el manejo transaccional será transparente.

Dentro de la aplicación que se expone, se hace el manejo de una librería que contiene varias clases y funciones:

Clase clbase

Contiene objetos y funcinoes comunes en la librería, será la clase padre. Hereda de la Clase modelo de componenetes de la biblioteca de clases del *Framework System.ComponentModel.Component*.

Objetos:

Friend Base As System.Data.OleDb.OleDbConnection
Friend Adaptador As System.Data.OleDb.OleDbDataAdapter
Friend SQLDelete As System.Data.OleDb.OleDbCommand
Friend SQLSelect As System.Data.OleDb.OleDbCommand
Friend SQLUpdate As System.Data.OleDb.OleDbCommand

Funciones:

Friend Sub AbrirBaseDatos*()*

Friend Sub CerrarBaseDatos*()*

Public Function FechaAct*() as Date*

Clase cursosaprobados

Esta clase sirve para mostrar funciones relacionadas con cursos aprobados de los estudiantes, y hereda funciones y objetos de la clase ClBase.

Funciones:

Public Function CursosAprob *(ByVal* Carnet *As String) As DataSet*

Public Function Promedio *(ByVal* Carnet *As String) As Double*

Public Function Creditos *(ByVal* Carnet *As String) As Integer*

Clase autenticación

Esta clase permite el manejo de los datos para validar la autenticación adecuada de usuarios al sistema, así como su nivel de permisos. También hereda funcones de la clase ClBase.

Funciones:

Public Function UsuarioValido*(ByVal* Carnet *As String, ByVal Password As String) As String*

Clase pensum

Esta clase permita la visualización de la inforamción relacionada con los pensum de las diferentes carreras relacionadas. Utiliza funciones y objetos heredados de la clase ClBase.

Funciones:

Public Function PensumCarrera*(ByVal* Cod_Carrera *As Integer) As DataSet*

Public Function Carreras*() As DataSet*

Clase asignación

Esta clase permite el manejo de las asiganciones de cursos para los estudiantes, también utiliza la herencia de la clase ClBase.

Eventos:

Estos eventos son funciones que no pueden ser llamadas desde un objeto que utilice la clase, sino son declaraciones para funciones que pueden dispararse al ejecutarse una acción dentro de la funcionalidad de una funcion en la clase, permitiendo así que se configure una función que ejecuta acciones específicas en el objeto que está instanciando la clase al momento de dispararse.

Public Event AsigError *(ByVal* Mensaje *As String)*

Public Event YaAsignado *(ByVal* Mensaje *as String)*

Public Event ConflictoHorario *(ByVal* Mensaje *as String)*

Funciones:

Public Function Semestre*() As String*

Public Function Boleta *(ByVal* Carnet *As String) As DataSet*

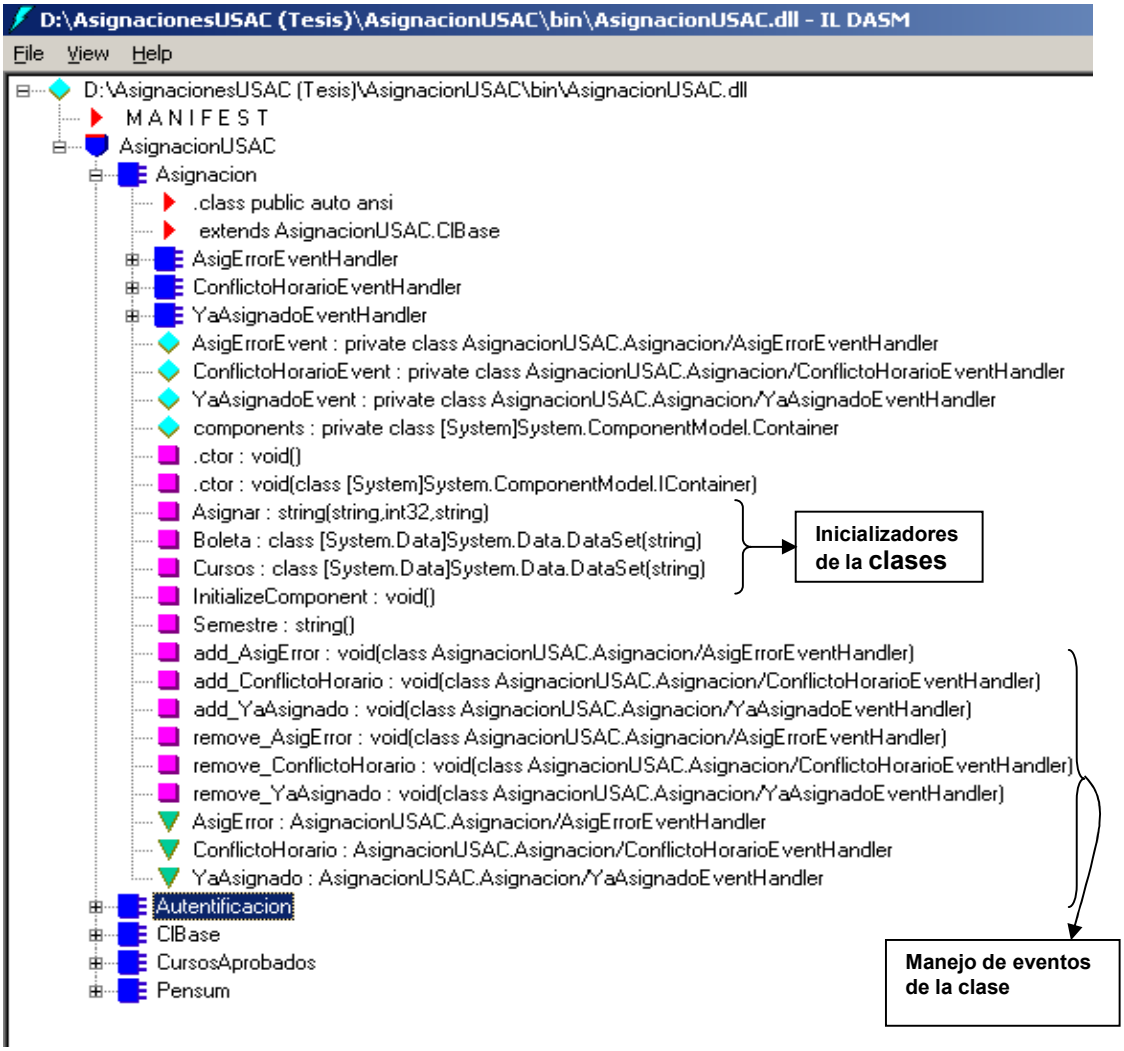
Public Function Cursos *(ByVal* Carnet *As String) As DataSet*

Public Function Asignar *(ByVal* Carnet *As String, ByVal* Cod_Curso *As Integer, ByVal* Seccion *As String) As String*

5.2.2.1 IL *Assembler* de las clases utilizadas

Es interesante visualizar la estructuración del lenguaje intermedio del .Net, lo cual se puede ver luego de ejecutar un desensamblador, donde se puede visualizar en la estructura el manejo de las clases, y ver la forma en que este lenguaje mantiene una estructuración orientada a objetos.

Figura 23. Códido desensamblado de aplicación



Debido a lo extendido que es el código de las funciones utilizadas en este sistema de ejemplificación y que el objetivo de este ejemplo es principalmente mostrar los resultados de la tecnología y no listar código con estructura definida y repetitiva, solamente visualizaremos un segmento de datos del IL, siendo un segmento que nos permita ver las generalidades del lenguaje

```
.assembly extern System.Data
{  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 1:0:2411:0
}
.assembly extern System.Xml
{  .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )
  .ver 1:0:2411:0
}
.assembly AsignacionUSAC
{  .custom instance void
   …….
  .hash algorithm 0x00008004
  .ver 1:0:892:4493
}
.mresource public AsignacionUSAC.Autentificacion.resources
{}
.mresource public AsignacionUSAC.dll.licenses
.module AsignacionUSAC.dll
.imagebase 0x11000000
.subsystem 0x00000002
.file alignment 4096
.corflags 0x00000001
.namespace AsignacionUSAC
{  .class public auto ansi CursosAprobados
        extends AsignacionUSAC.ClBase
   {    .field private class [System]System.ComponentModel.Container
components
```

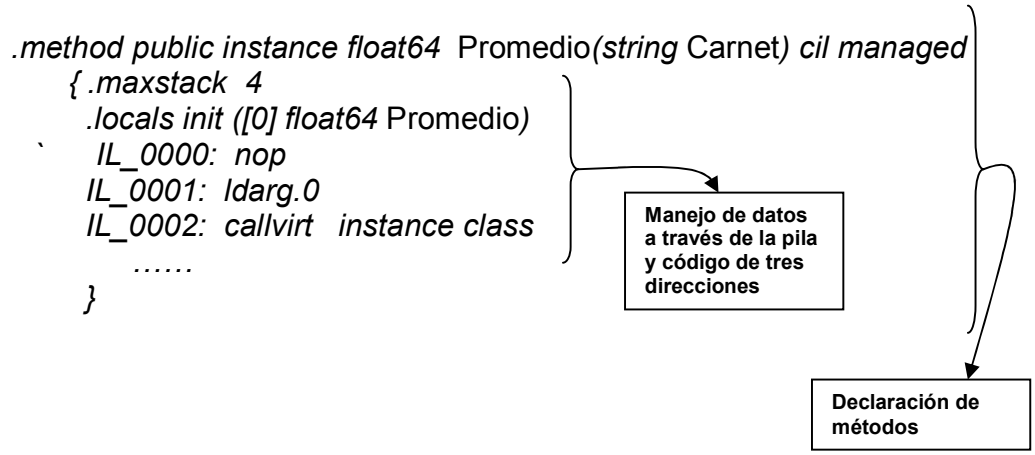
Manejo de ensamblados relacionados

Declaración del ensamblado principal

Declaración de recursos

Declaración del espacio de nombres

Declaración de una clase en el espacio de nombres y herencia de objetos a través de las extensiones.



5.2.3 Capa de servicios de datos

Esta capa permite la publicación de la información que se maneja, basandose en el estandar SOAP, para la transmisión de datos en un formato XML. Es importante mencionar que el tener un capa de manejo de datos de esta forma, permite que los mismos sean compartidos con otros sistemas u otras páginas, sin necesidad de que la página sea diseñada estrictamente en ASP.Net, esto ayuda a dar apertura al manejo de la información.

Los datos son publicados de tal forma que se pueden manejar con objetos de almacenamiento de información, sin embargo, en necesario que los datos sean serializables, es decir convertibles a un formato XML en forma lógica y coherente para su interpretación.

Los servicios *web*, son creados como clases de igual forma que una librería o un formulario, con la diferencia que estas clases heredan datos de la clase *System.Web.Services.WebService*, asimismo se debe colocar el texto <WebMethod()> antes de las funciones.

En el caso de la aplicación planteada, se publicó un clases dentro del servicio *web* para cada clase de la capa de lógica de funcionalidad, ya que lo que se pretende es simplemente publicar la información y las funciones que se pueden obtener a través de las clases mencionadas.

5.2.3.1 Ejecución y carga del lado del servidor

Debido a que en esta sección lo que se desea es únicamente poder ejemplificar como visualizar el resultado de una ejecución de un servicio *Web*, el cual retorna una tabla de datos, solamente mostraremos el resultado de la ejecución del servicio de **Cursos Aprobados**, enviando como parámetro del número de carnet **9616938**, quedando un archivo XML en el cual se puede ver perfectamente la estructura de la tabla resultado, lo cual esta basado en el anidamiento de etiquetas, tal como se explico en el capítulo.

```
<?xml version="1.0" encoding="utf-8" ?>
<DataSet xmlns="http://tempuri.org/">
<xsd:schema id="NewDataSet" targetNamespace="" xmlns=""
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
        <xsd:element name="NewDataSet"
  msdata:IsDataSet="true">
          <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
              <xsd:element name="Table">
                <xsd:complexType>
                  <xsd:sequence>
                      <xsd:element
                name="CODIGO"
                type="xsd:decimal" minOccurs="0"
                />
                      <xsd:element
                name="NOMBRE"
                type="xsd:string" minOccurs="0" />
                .....
```

Encabezado del XML generado

Esquema del los datos contenidos, es decir los metadatos del resultado

```
                    <xsd:element name="NOTA"
                    type="xsd:decimal" minOccurs="0"
                    />
                       </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:choice>
              </xsd:complexType>
            </xsd:element>
          </xsd:schema>
           <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-
com:xml-msdata" xmlns:diffgr="urn:schemas-microsoft-
com:xml-diffgram-v1">
            <NewDataSet xmlns="">
              <Table diffgr:id="Table1" msdata:rowOrder="1">
                 <CODIGO>25</CODIGO>
                 <NOMBRE>Practicas rimarias</NOMBRE>
                 <CREDITOS>3</CREDITOS>
                <APROB>30/06/1996</APROB>
                 <ZONA>45</ZONA>
                 <FINAL>25</FINAL>
                 <NOTA>70</NOTA>
              </Table>
              <Table diffgr:id="Table3" msdata:rowOrder="2">
                    <CODIGO>39</CODIGO>
                    <NOMBRE>Deportes 1</NOMBRE>
                    <CREDITOS>1</CREDITOS>
                    <APROB>30/06/1996</APROB>
                    <ZONA>45</ZONA>
                    <FINAL>25</FINAL>
                    <NOTA>70</NOTA>
              </Table>
            </NewDataSet>
          </diffgr:diffgram>
        </DataSet>
```

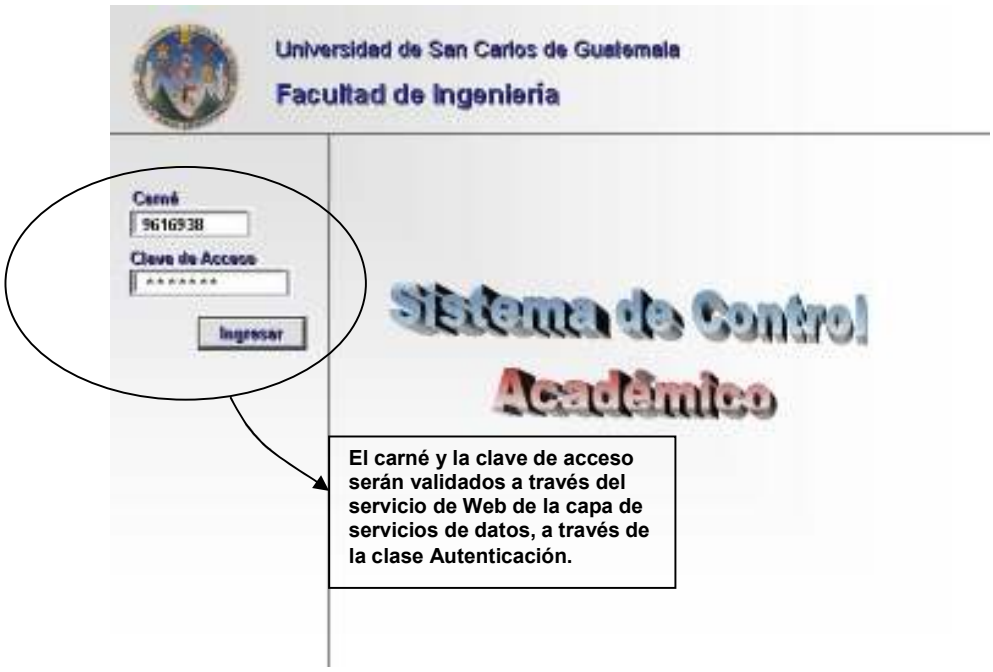
Información de una Tupla contenida en el Resultado, enmarcada por cada una de sus columnas.

5.2.4 Capa de presentación

Esta capa es importante debido a que en ella es donde se refleja toda la funcionalidad de fondo del sistema. La diferencia entre aplicaciones ASP.Net y las ASP tradicionales, es que las aplicaciones ASP solamente pueden incluir scripts que tienen un nivel de funcionalidad amplio para las páginas, sin embargo los ASP.Net permiten tomar todo el poder el lenguaje y no solo del segmento de scripts, esto debido a que la página no es interpretada, sino es ejecutada bajo un código compilado, el cual es administrado en forma adecuada por el *Framework* del lado del servidor, lo cual permite el manejo de datos sin instalaciones de componentes del lado del cliente.

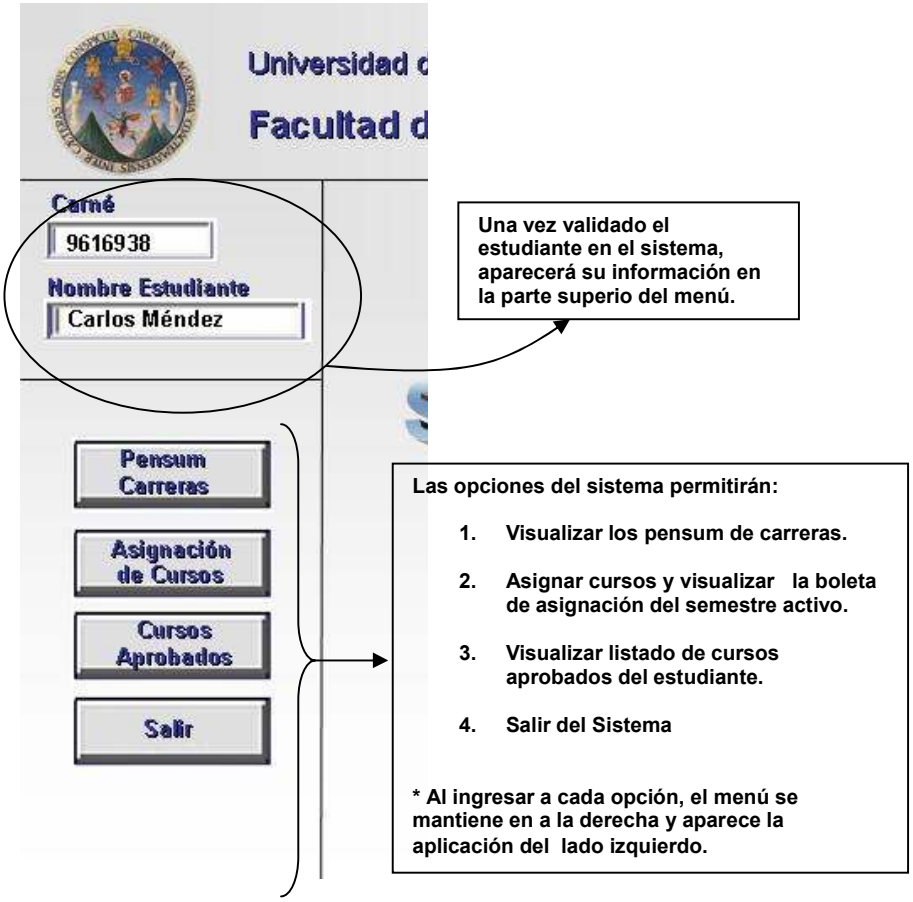
En la parte funcional es posible observar la pantalla de ingreso, desde la cual se será necesario el ingreso de un carné y una clave de acceso, lo cual será validado para habilitar el menú del sistema con las opciones válidas.

Figura 24. Pantalla de ingreso al sistema



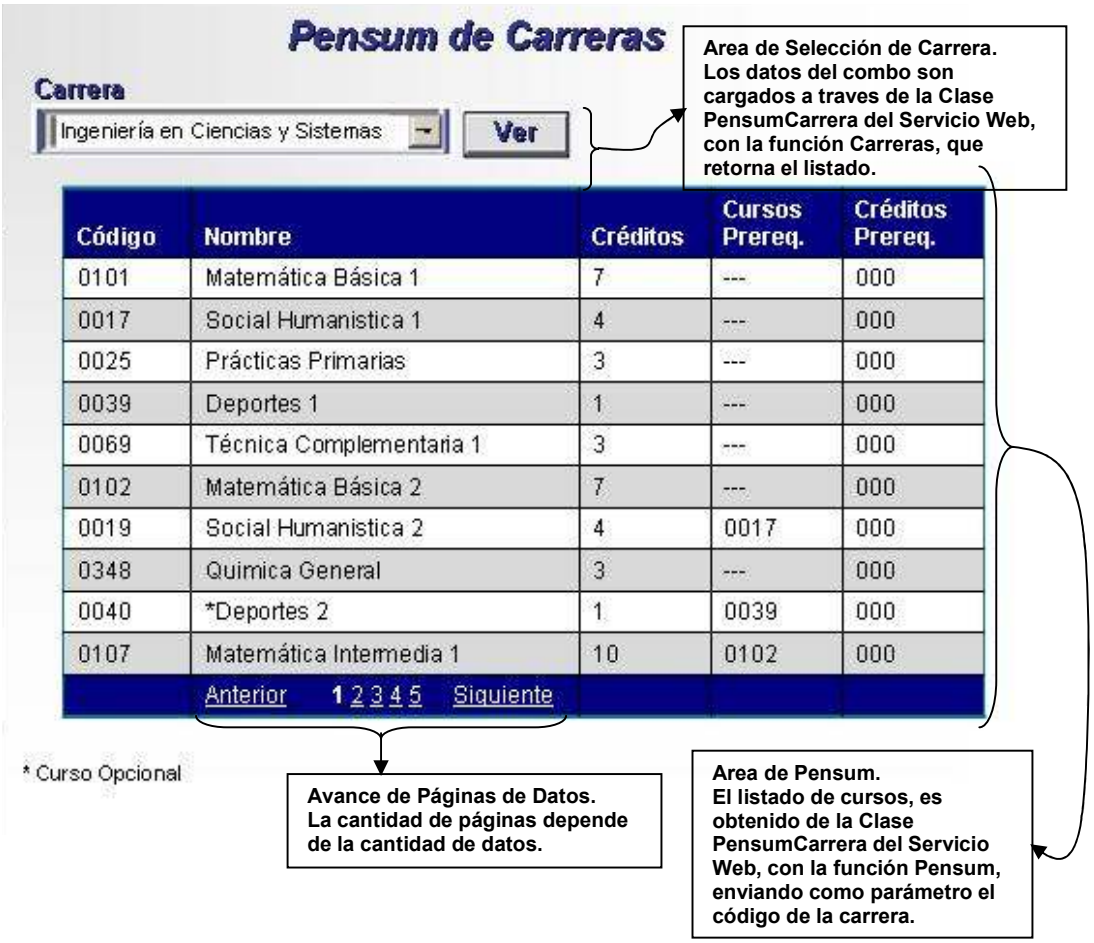
Al estar validado en forma adecuada el estudiante pordrá observar la pantalla de menú, con las diversas funciones son habilitados, de igual forma internamente el sistema iniciará el manejo de la sesión del estudiante dentro del sistema.

Figura 25. Pantalla menú del sistema



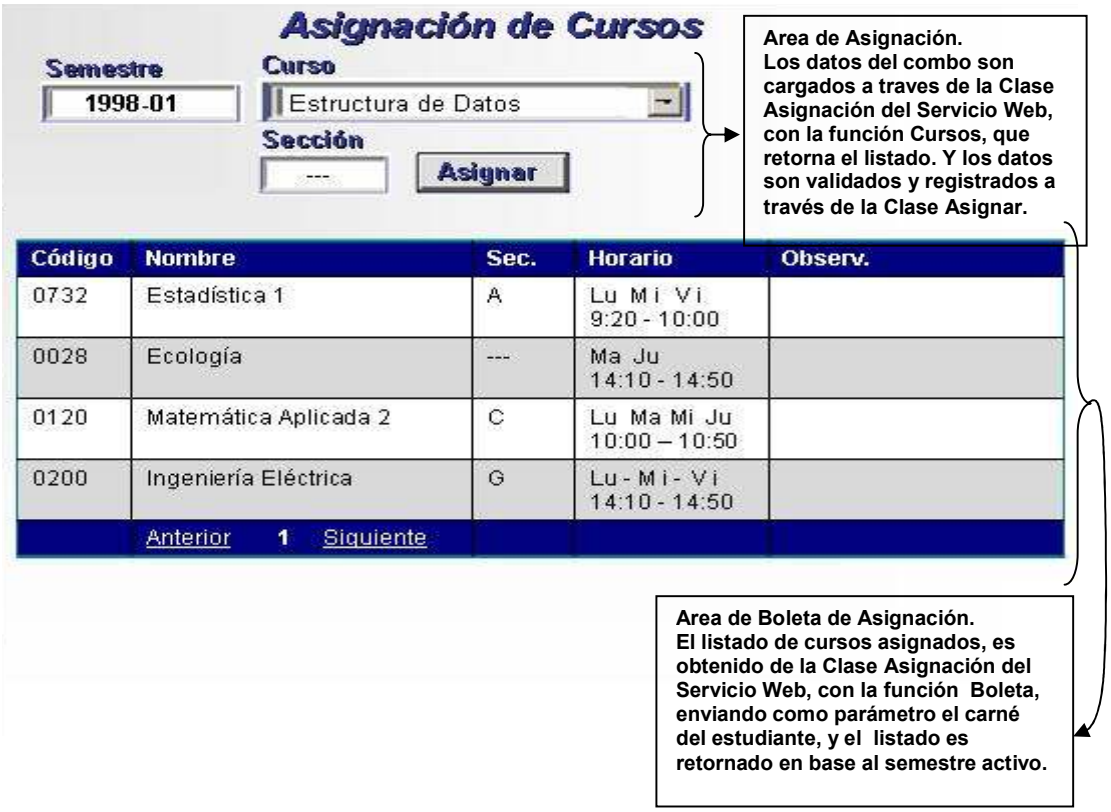
Al utilizar la opción de **pensum carreras**, se podrá la aplicación de dicha funcionalidad, la cual permite la selección una carrera específica y luego al presionar el botón **ver**, se mostrará el listado de cursos de la carrera, incluyendo su código, nombre, créditos y los prerequisitos en cursos aprobados y creditos necesarios para optar por el curso.

Figura 26. Pantalla pensum de carreras



Al utilizar la opción de **asignación de cursos**, se podrá observar una pantalla donde se podrán ingresar datos para asignar un curso nuevo para el estudiante en el semestre activo (parametrización del sistema), desde el momento de ingreso a la opción se visualiza el listado de cursos ya asignados, y esta se irá modificando cada vez que se asigna otro curso. El listado incluye para cada curso su código, nombre, créditos, sección, horario y observaciones en caso de cursos problema.

Figura 27. Pantalla asignación de cursos



Finalmente la última opción es la de **cursos aprobados**, la cual muestra el listado de cursos aprobados del estudiante validado en el sistema. La clase del servicio Web que alimenta este formulario fue mostrado en la sección anterior, mostrandola en formato XML como el resultado del servicio Web, ahora será mostraremos a través de una página ASP.Net, cuya obtención de datos solamente consiste en solicitarlos y cargarlos a un objeto *DataGrid* a través de la función del *Framework* llamada *DataBind*, El listado incluye para cada curso su código, nombre, créditos, fecha de aprobación y nota final del curso.

Figura 28. Pantalla cursos aprobados

CONCLUSIONES

1. Las capacidades generales fundamentadas en la nueva visión de desarrollo sustentada en tecnología .Net, permiten un nivel de programación complejo en resultados pero muy práctico en estructura y elaboración.

2. Como se vio a lo largo de los diversos capítulos y como se pudo ejemplificar en el último, la estructura totalmente orientada a objetos le permite a los lenguajes .Net centralizarse en un eje que permite la interacción entre lenguajes, a través de un lenguaje intermedio al cual se compilan los programas.

3. El rendimiento de desarrollo de aplicaciones complejas, puede ser ampliamente minimizado utilizando .Net, pues muchas cosas genéricas como es el manejo de datos en tablas de presentación o manejo de ventanas ya está proporcionado en las bibliotecas del *Framework*, permitiendo con esto que la programación se oriente más al objetivo real, la lógica del proceso.

4. A lo largo del trabajo de graduación y en el ejemplo final, la utilización del XML es un punto clave y muy importante en esta tecnología, pues este junto con el protocolo SOAP permiten hacer el manejo complejo de datos, pues el ADO.Net queda totalmente orientado a manejar los datos de esta forma.

5. La capacidad de utilizar los servicios *web*, nos permite aumentar el manejo de crecimiento en capas, como lo vimos en el ejemplo, lo cual permite que se pueda tener una mayor escalabilidad en los diversos sistemas.

6. La generación de aplicaciones totalmente orientadas a objetos, en el ensamblador, permite la transparencia en la utilización de codigo fuente, permitiendo la interacción de lenguajes diversos y manteniendo las características de los objetos, tales como encriptamiento, reutlización de código.

7. La utilización del ASP.Net para el desarrollo de aplicaciones *web*, permite el manejo de capas de sistemas, aprovechando los beneficios de utilizar los servicios *web*, clases y objetos con funcionalidades del lado del servidor, lo cual permite aplicaciones más livianas del lado del cliente.

8. La utilización de librerias DLL en una capa de lógica de negocio en un sitio desarrollado para .Net, no está sujeto a registro de clases, facilitando su actualización, versionamiento y opraciones independientes entre sitios o páginas en el mismo servidor.

RECOMENDACIONES

1. Para desarrollar aplicaciones con .Net, es importante tomar en cuenta la capa de clases en el servidor, que puede ser operativa tanto para aplicaciones *web* como aplicaciones *Windows*, aprovechando la reutilización de código y manteniendo el encriptamiento del mismo.

2. Al momento de diseñar aplicaciones *web*, es importante tomar en cuenta la capacidad del manejo de sesiones y la transportación de datos a través de objetos serializables (es decir, transformables en XML), ya que esto ayuda al paso de datos de un formulario a otro sin utilizar el envio de parámetros.

3. Las aplicaciones ASP.Net pueden llegar a ser casi tan complejas como una aplicación Windows, pudiendose utilizar objetos y programas más completos que con los *VB Scripts o Java Scripts*, por lo que se debe poner enfasis en la carga del lado del servidor, pero tamando en cuenta que si es posible el uso de *ActiveX* o *Java Applets* en el lado del cliente y que la capacidad de estos puede aumentarse través de utilizar código .Net (requiere el *Framework* del lado cliente).

4. Los servicios *web* pueden ser muy complejos o muy simples, sin embargo por seguridad de código, lo recomenbable es utilizar servicios simples, que contengan únicamente llamadas a librerias DLL, que contengan la complejidad de la funcionalidad, convirtiendose el servicio *web* en un medio de transporte de datos, permitiendo la escalabilidad de la funcionalidad sin alterar los servicios.

BIBLIOGRAFÍA

Lidin Serge, ***Incide Microsoft .Net IL Assembler***, Microsoft *Press*, febrero 6 de 2002.

Microsoft *Team*, ***.Net Framework SKD Documentation***, Microsoft *Tech Net*, agosto de 2001.

Platt David S., Ballinger Keith, **Microsoft *.Net Introducing***, Microsoft *Press*, 1era. edición, mayo 16 del 2001.

Reilly Douglas J., **Microsoft ASP.Net *Aplications***, Microsoft *Press*, noviembre 12 de 2001.

Richter Jerry, ***Applied* Microsoft .Net *Framework Programming***, Microsoft *Press*, 1era. edición, enero 23 de 2002.

Visual Studio Magazine, volumen 12, No. 4, abril de 2002.

MSDN *Magazine*, volumen 17, No. 4, abril de 2002.

ANEXOS

Tabla XVIII. Esquema de tipos comunes que puede manejar el .Net

Categoría	Clase	Descripción	VB	C#	C++
Entero	*byte*	Sin signo de 8-bits.	*byte*	*byte*	*char*
	sbyte	Con signo de 8-bits.	*sbyte*	*sbyte*	*signed char*
	int16	Con signo de 16-bits.	*short*	*short*	*short*
	int32	Con signo de 32-bits.	*integer*	*int*	*int or long*
	int64	Con signo de 64-bits.	*long*	*long*	*__int64*
	uint16	Sin signo de 16-bits.	*uint16*	*ushort*	*short*
	uint32	Sin signo de 32-bits.	*uint32*	*uint*	*unsigned int*
	uint64	Sin signo de 64-bits.	*uint64*	*ulong*	*__int64*
Punto flotante	*single*	Precisión Simple (32-bit)	*single*	*float*	*float*
	double	Precisión Simple (64-bit)	*double*	*double*	*double*
Lógicos	*boolean*	Valor Booleano	*boolean*	*bool*	*bool*
Otros	*char*	Caracter (16-bit).	*char*	*char*	*wchar_t*
	decimal	Valor decimal de 96-bits.	*decimal*	*decimal*	*decimal*
	intptr	Con signo de tamaño variable.	*intptr*	*intptr*	*intptr*
Otros	*uintptr*	Sin signo de tamaño vairable.	*uintptr*	*uintptr*	*uintptr*
Clases	*object*	Ruta de objeto heredado.	*object*	*object*	*object**
	string	Inmutable, string de con largo fijo, o Unicode.	*string*	*string*	*string**

Richter Jerry, *Applied* Microsoft .Net *Framework Programming*, Estados Unidos de Norte América, pp. 164

Tabla XIX. Instrucciones del lenguaje intermedio

Código	Nombre	Parámetros	Pop (Pila)	Push (Pila)
00	nop	-	-	-
01	break	-	-	-
02	ldarg.0	-	-	*
03	ldarg.1	-	-	*
04	ldarg.2	-	-	*
05	ldarg.3	-	-	*
06	ldloc.0	-	-	*
07	ldloc.1	-	-	*
08	ldloc.2	-	-	*
09	ldloc.3	-	-	*
0A	stloc.0	-	*	-
0B	stloc.1	-	*	-
0C	stloc.2	-	*	-
0D	stloc.3	-	*	-
0E	ldarg.s	uint8	-	*
0F	ldarga.s	uint8	-	&
10	starg.s	uint8	*	-
11	ldloc.s	uint8	-	*
12	ldloca.s	uint8	-	&
13	stloc.s	uint8	*	-
14	ldnull	-	-	&=0
15	ldc.i4.m1ldc.i4.M1	-	-	int32=-1
16	ldc.i4.0	-	-	int32=0
17	ldc.i4.1	-	-	int32=1
18	ldc.i4.2	-	-	int32=2
19	ldc.i4.3	-	-	int32=3
1A	ldc.i4.4	-	-	int32=4
1B	ldc.i4.5	-	-	int32=5
1C	ldc.i4.6	-	-	int32=6
1D	ldc.i4.7	-	-	int32=7
1E	ldc.i4.8	-	-	int32=8
1F	ldc.i4.s	int8	-	int32
20	ldc.i4	int32	-	int32
21	ldc.i8	int64	-	int64
22	ldc.r4	float32	-	Float

Lidin Serge, *Incide Microsoft .Net IL Assembler,* Estados Unidos de Norte América, pp. 420 - 427

Continuación

Código	Nombre	Parámetros	Pop (Pila)	Push (Pila)
23	ldc.r8	float64	-	Float
25	dup	-	*	*,*
26	Pop	-	*	-
27	jmp	<Método>	-	-
28	call	<Método>	N argumentos	Val. Ret.
29	calli	<Firma>	N argumentos	Val. Ret.
2ª	ret	-	*	-
2B	br.s	int8	-	-
2C	brfalse.sbrnull.sbrzero.s	int8	int32	-
2D	brtrue.sbrinst.s	int8	int32	-
2E	beq.s	int8	*,*	-
2F	bge.s	int8	*,*	-
30	bgt.s	int8	*,*	-
31	ble.s	int8	*,*	-
32	blt.s	int8	*,*	-
33	bne.un.s	int8	*,*	-
34	bge.un.s	int8	*,*	-
35	bgt.un.s	int8	*,*	-
36	ble.un.s	int8	*,*	-
37	blt.un.s	int8	*,*	-
38	br	int32	-	-
39	brfalsebrnullbrzero	int32	int32	-
3A	brtruebrinst	int32	int32	-
3B	beq	int32	*,*	-
3C	bge	int32	*,*	-
3D	bgt	int32	*,*	-
3E	ble	int32	*,*	-
3F	blt	int32	*,*	-
40	bne.un	int32	*,*	-
41	bge.un	int32	*,*	-
42	bgt.un	int32	*,*	-
43	ble.un	int32	*,*	-
44	blt.un	int32	*,*	-
45	switch	(uint32=N) + N(int32)	*,*	-
46	ldind.i1	-	&	int32

Lidin Serge, *Incide Microsoft .Net IL Assembler,* Estados Unidos de Norte América, pp. 420 - 427

Continuación

Código	Nombre	Parámetros	Pop (Pila)	Push (Pila)
47	ldind.u1	-	&	int32
48	ldind.i2	-	&	int32
49	ldind.u2	-	&	int32
4A	ldind.i4	-	&	int32
4B	ldind.u4	-	&	int32
4C	ldind.i8ldind.u8	-	&	int64
4D	ldind.i	-	&	int32
4E	ldind.r4	-	&	Float
4F	ldind.r8	-	&	Float
50	ldind.ref	-	&	&
51	stind.ref	-	&,&	-
52	stind.i1	-	int32,&	-
53	stind.i2	-	int32,&	-
54	stind.i4	-	int32,&	-
55	stind.i8	-	int32,&	-
56	stind.r4	-	Float,&	-
57	stind.r8	-	Float,&	-
58	add	-	*,*	*
59	sub	-	*,*	*
5A	mul	-	*,*	*
5B	div	-	*,*	*
5C	div.un	-	*,*	*
5D	rem	-	*,*	*
5E	rem.un	-	*,*	*
5F	and	-	*,*	*
60	or	-	*,*	*
61	xor	-	*,*	*
62	shl	-	*,*	*
63	shr	-	*,*	*
64	shr.un	-	*,*	*
65	neg	-	*	*
66	not	-	*	*
67	conv.i1	-	*	int32
68	conv.i2	-	*	int32
69	conv.i4	-	*	int32
6A	conv.i8	-	*	int64

Lidin Serge, *Incide Microsoft .Net IL Assembler,* Estados Unidos de Norte América, pp. 420 - 427

Continuación

Código	Nombre	Parámetros	Pop (Pila)	Push (Pila)
6B	conv.r4	-	*	Flota
6C	conv.r8	-	*	Flota
6D	conv.u4	-	*	int32
6E	conv.u8	-	*	int64
6F	callvirt	<Método>	N argumentos	Val. Ret.
70	cpobj	<Tipo>	&,&	-
71	ldobj	<Tipo>	&	*
72	ldstr	<String>	-	O
73	newobj	<Método>	N argumentos	O
74	castclass	<Tipo>	o	O
75	isinst	<Tipo>	o	int32
76	conv.r.un	-	*	Float
79	unbox	<Tipo>	o	&
7A	throw	-	o	-
7B	ldfld	<Campo>	o/&	*
7C	ldflda	<Campo>	o/&	&
7D	stfld	<Campo>	o/&,*	-
7E	ldsfld	<Campo>	-	*
7F	ldsflda	<Campo>	-	&
80	stsfld	<Campo>	*	-
81	stobj	<Tipo>	&,*	-
82	conv.ovf.i1.un	-	*	int32
83	conv.ovf.i2.un	-	*	int32
84	conv.ovf.i4.un	-	*	int32
85	conv.ovf.i8.un	-	*	int64
86	conv.ovf.u1.un	-	*	int32
87	conv.ovf.u2.un	-	*	int32
88	conv.ovf.u4.un	-	*	int32
89	conv.ovf.u8.un	-	*	int64
8A	conv.ovf.i.un	-	*	int32
8B	conv.ovf.u.un	-	*	int64
8C	box	<Tipo>	*	o
8D	newarr	<Tipo>	int32	o
8E	ldlen	-	o	int32
8F	ldelema	<Tipo>	int32,o	&
90	ldelem.i1	-	int32,o	int32

Lidin Serge, *Incide Microsoft .Net IL Assembler,* Estados Unidos de Norte América, pp. 420 - 427

Continuación

Código	Nombre	Parámetros	Pop (Pila)	Push (Pila)
91	ldelem.u1	-	int32,o	int32
92	ldelem.i2	-	int32,o	int32
93	ldelem.u2	-	int32,o	int32
94	ldelem.i4	-	int32,o	int32
95	ldelem.u4	-	int32,o	int32
96	ldelem.i8ldelem.u8	-	int32,o	int64
97	ldelem.i	-	int32,o	int32
98	ldelem.r4	-	int32,o	Float
99	ldelem.r8	-	int32,o	Float
9A	ldelem.ref	-	int32,o	o/&
9B	stelem.i	-	int32,int32,o	-
9C	stelem.i1	-	int32,int32,o	-
9D	stelem.i2	-	int32,int32,o	-
9E	stelem.i4	-	int32,int32,o	-
9F	stelem.i8	-	int64,int32,o	-
A0	stelem.r4	-	Float,int32,o	-
A1	stelem.r8	-	Float,int32,o	-
A2	stelem.ref	-	o/&,int32,o	-
B3	conv.ovf.i1	-	*	int32
B4	conv.ovf.u1	-	*	int32
B5	conv.ovf.i2	-	*	int32
B6	conv.ovf.u2	-	*	int32
B7	conv.ovf.i4	-	*	int32
B8	conv.ovf.u4	-	*	int32
B9	conv.ovf.i8	-	*	int64
BA	conv.ovf.u8	-	*	int64
C2	refanyval	<Tipo>	*	&
C3	ckfinite	-	*	Float
C6	mkrefany	<Tipo>	&	*
D0	ldtoken	<Tp/cp/mt>	-	&
D1	conv.u2	-	*	int32
D2	conv.u1	-	*	int32
D3	conv.i	-	*	int32
D4	conv.ovf.i	-	*	int32
D5	conv.ovf.u	-	*	int32
D6	add.ovf	-	*,*	*

Lidin Serge, *Incide Microsoft .Net IL Assembler,* Estados Unidos de Norte América, pp. 420 - 427

Continuación

Código	Nombre	Parámetros	Pop (Pila)	Push (Pila)
D7	add.ovf.un	-	*,*	*
D8	mul.ovf	-	*,*	*
D9	mul.ovf.un	-	*,*	*
DA	sub.ovf	-	*,*	*
DB	sub.ovf.un	-	*,*	*
DC	endfinallyendfault	-	-	-
DD	leave	int32	-	-
DE	leave.s	int8	-	-
DF	stind.i	-	int32,&	-
E0	conv.u	-	*	int32
FE 00	arglist	-	*	&
FE 01	ceq	-	*,*	int32
FE 02	cgt	-	*,*	int32
FE 03	cgt.un	-	*,*	int32
FE 04	clt	-	*,*	int32
FE 05	clt.un	-	*,*	int32
FE 06	ldftn	<Método>	-	&
FE 07	ldvirtftn	<Método>	o	&
FE 09	ldarg	uint32	-	*
FE 0A	ldarga	uint32	-	&
FE 0B	starg	uint32	*	-
FE 0C	ldloc	uint32	-	*
FE 0D	ldloca	uint32	-	&
FE 0E	stloc	uint32	*	-
FE 0F	localloc	-	int32	&
FE 11	endfilter	-	int32	-
FE 12	unaligned.	uint8	-	-
FE 13	volatile.	-	-	-
FE 14	tail.	-	-	-
FE 15	initobj	<Tipo>	&	-
FE 17	cpblk	-	int32,&,&	-
FE 18	initblk	-	int32,int32,&	-
FE 1A	rethrow	-	-	-
FE 1C	sizeof	<Tipo>	-	int32
FE 1D	refanytype	-	*	

Lidin Serge, *Incide Microsoft .Net IL Assembler,* Estados Unidos de Norte América, pp. 420 - 427