UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

TESIS PREVIA A LA OBTENCIÓN DEL TÍTULO DE: INGENIERO EN SISTEMAS COMPUTACIONALES

TEMA:

IMPLEMENTACIÓN DE APLICACIONES .NET PARA PLATAFORMA DE DESARROLLO LIBRE, BASADA EN LINUX Y COMPATIBLE CON MICROSOFT.NET.

APLICATIVO:

PORTAL PARA EL CENTRO DE CAPACITACIÓN CONTINUA DE LA FICA

AUTOR: PABLO ANDRÉS LANDETA LÓPEZ

DIRECTOR: ING. IRVING REASCOS

FEBRERO 2004

IBARRA – ECUADOR

CERTIFICACIÓN

Certifico que esta tesis ha sido elaborada en su totalidad por el Egresado Pablo Andrés Landeta López bajo mi dirección y asesoramiento.

……

Ing. Irving Reascos

DIRECTOR DE TESIS

AGRADECIMIENTO

A todas las personas que forman parte de la Escuela de Ingeniería en Sistemas Computacionales, en especial a los maestros que depositaron en mí todos los conocimientos necesarios para formarme como profesional.

Al Ing. Irving Reascos, Director de Tesis del presente proyecto, que con sus amplios conocimientos supo guiarme en la realización de este trabajo de investigación.

A mis padres, ya que gracias a su total apoyo moral y económico, culmino con éxito mis estudios universitarios.

DEDICATORIA

A mis padres, quienes en pos de mi superación, sembraron el anhelo de luchar por mis ideales guiándome en el camino del sacrificio, la fortaleza y la abnegación; gracias por todo el esfuerzo, amor y cariño que día a día me han brindado ayudándome a crecer como persona.

Pablo Andrés Landeta López

INDICE

ANEXOS EN EL CD:

\\Aplicativo
Contiene todo el aplicativo

\\Sitio Web
Contiene todas las páginas web que forman el aplicativo

\\Base de datos
Contiene la base de datos física del aplicativo

\\Contenido
Contiene el presente documento

\\Diapositivas
Contiene las presentaciones de la defensa práctica

\\Instaladores
Contiene los instaladores del software para que funcione el sistema

\\Mono 0.28
Contiene la versión 0.28 de Mono, utilizada para la creación del portal del Centro de Capacitación Continua

\\Mono 1.0
Contiene la última versión de Mono

\\Manuales
Contiene los manuales de usuario, técnico y de instalación de los diferentes componentes para que el sistema opere en forma normal

INTRODUCCIÓN

.NET es una plataforma de software, es un ambiente independiente del lenguaje para escribir programas que pueden ser fáciles y seguros de interoperar. Más que dirigirse a una combinación de hardware – sistema operativo en particular, los programas se dirigen a .NET, y corren en cualquier lugar que .NET esté implementado.

.NET también es un nombre colectivo dado a varias partes de software construidos sobre la plataforma .NET.

En el Capítulo I se estudia a .NET como tecnología creada por Microsoft como una propuesta de innovación tecnológica.

En el Capítulo II se estudia a Mono, el cual es un paquete de software que se constituye en una implementación de la plataforma de desarrollo .NET para el sistema operativo Linux.

En el Capítulo III se estudia a C# como un lenguaje de programación que presenta grandes características al utilizar como base la programación orientada a objetos.

En el Capítulo IV se estudia a ADO.NET, el cual es una parte de la tecnología .NET para el acceso a base de datos.

En el Capítulo V se estudia a ASP.NET, que es la tecnología de .NET para el desarrollo de aplicaciones basadas en Web.

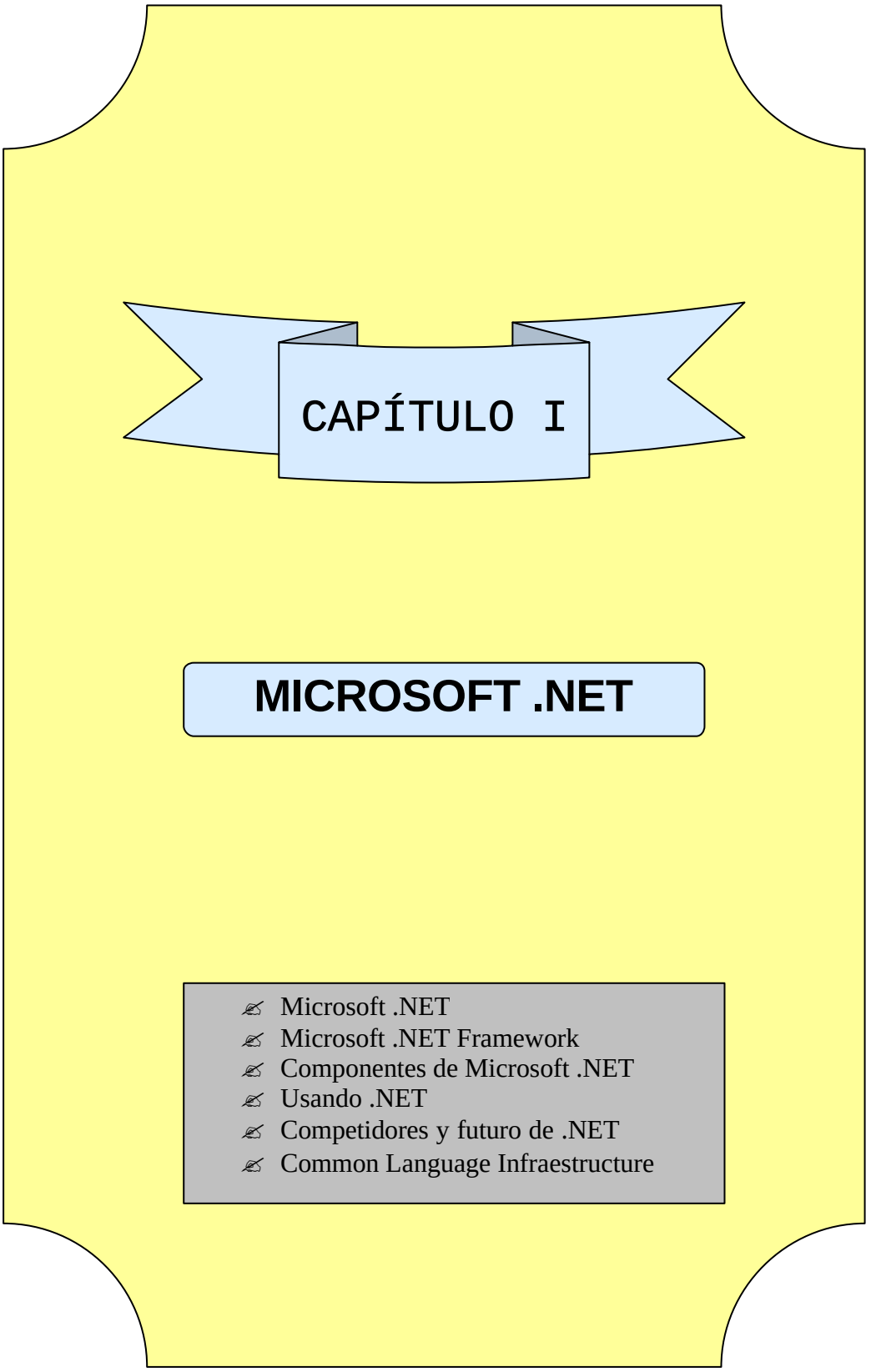
En el Capítulo VI se estudia a los servicios Web XML, que en realidad son una parte de ASP.NET, pero por su gran importancia y magnitud, se los ha separado del Capítulo V para tratarlo en un capítulo individual.

En el Capítulo VII se explica la realización del Aplicativo, el cual es el Portal para el centro de Capacitación Continua de la FICA, realizado íntegramente en Linux con la tecnología Mono y que expone la genialidad de .NET en el desarrollo de aplicaciones basadas en Web.

En el Capítulo VIII, se presenta la verificación de la hipótesis y se presenta un conjunto de conclusiones y recomendaciones acerca del presente trabajo investigativo.

Por último se presenta como Anexos al Anteproyecto de Tesis, de manera que se pueda observar la idea inicial previa al trabajo que se está exponiendo como Tesis de Grado.

En los 2 CD ROM adjuntos a la presente tesis se incluyen otros anexos, como son: el aplicativo, los instaladores, el contenido, las diapositivas y los manuales.


CAPÍTULO I
MICROSOFT .NET
Microsoft .NET
Microsoft .NET Framework
Componentes de Microsoft .NET
Usando .NET
Competidores y futuro de .NET
Common Language Infraestructure

1.1 MICROSOFT .NET

.NET es la plataforma de Microsoft para el desarrollo, despliegue y ejecución de aplicaciones orientadas a servicios sobre entornos altamente distribuidos, tal y como lo es Internet. Además introduce el concepto de los servicios Web, que permiten el desarrollo de aplicaciones acopladas basadas en componentes que utilizan protocolos de comunicación estándares de Internet como SOAP (*Single Object Access Protocol*) y XML (*Extensible Markup Language*).

La plataforma .NET no es un sistema operativo, al menos por el momento, si bien está bastante integrada con este, y hace uso de los servicios que le proporciona. El estado actual de .NET podría compararse con el estado del entorno Windows 3.1 con respecto al sistema operativo MS-DOS(*Microsoft Disk Operating System*), por lo que es más que probable que la plataforma .NET acabe fundiéndose con una futura versión del sistema operativo Windows.

La plataforma .NET en realidad no es algo radicalmente nuevo. Es un conjunto de tecnologías dispersas, que en muchos casos ya existían, y que Microsoft ha integrado en una plataforma común con el objetivo de facilitar el desarrollo de este nuevo tipo de servicios de tercera generación.

.Net representa la visión del software como un servicio, habiendo sido diseñada con Internet en mente. Esta plataforma cubre todas las capas del desarrollo de software, existiendo una alta integración entre las tecnologías de presentación, de componentes y de acceso a datos. Intenta poner un cierto orden sobre el caos del desarrollo de aplicaciones distribuidas, que se basaba en un modelo de tres capas, con ASP(*Active Server Pages*) en la capa de

presentación, COM[1] en la capa de objetos de negocio y ADO(*Access Data Objects*) en la capa de datos; dicha plataforma tenía como problemas principales que el desarrollo con COM era complejo y poseía una integración con ASP un tanto artificiosa.

La plataforma .Net ha sido diseñada con la intención de satisfacer los siguientes objetivos:

- Proporcionar un modelo de programación simple y consistente. A diferencia de los modelos ya pasados, en los cuales algunas facilidades del sistema operativo son ofrecidas mediante DLL's y otras mediante objetos COM, todos los servicios de Mono son proporcionados de la misma forma mediante un modelo de programación orientado a objetos. Así mismo, se ha simplificado el modelo de programación, lo que permite a los desarrolladores centrarse en las cuestiones relativas a la lógica de la aplicación; se ha eliminado la necesidad de generar ficheros IDL, gestionar el registro, etc.
- Liberar al programador de las cuestiones de infraestructura (aspectos no funcionales). Así, .Net se encarga de gestionar automáticamente tales cuestiones como la gestión de la memoria, de los hilos o de los objetos remotos.
- Proporcionar integración entre diferentes lenguajes. Con el auge de los sistemas distribuidos, la interoperabilidad se ha convertido en una de las principales cuestiones de los desarrolladores de sistemas. El problema de la interoperabilidad ha sido considerado durante muchos años, desarrollándose varios estándares y arquitecturas como son los estándares arquitecturales (RPC – *Remote Procedure Calling*, CORBA - *Common Object Request Broker Architecture*, COM, los estándares de lenguajes ANSI C, etc.).
- Proporcionar una ejecución multiplataforma. .NET ha sido diseñado para ser independiente de la plataforma sobre la cual se ejecutarán las

[1] COM: provee la tecnología de Componentes para la arquitectura Microsoft Windows Distributed interNet Aplicactions (Windows DNA)

aplicaciones. Para conseguir este objetivo las aplicaciones .NET se compilan a un lenguaje intermedio denominado Lenguaje Común Intermedio (CIL, *Common Intermediate Language*), el cual es independiente de las instrucciones de una CPU concreta.

- Sistema de despliegue simple. Se ha eliminado la necesidad de tratar con el registro, con GUIDs (cadena de identificación única usada con llamadas a procedimientos remotos), etc., de forma que la instalación de una aplicación es tan sencilla como su copia en un directorio.
- Mejora de la escalabilidad. La gestión por parte del sistema de ejecución de .NET de cuestiones como la memoria permite mejorar la escalabilidad.
- Proporcionar un mecanismo de seguridad avanzado. El aumento de la dependencia sobre el código móvil, como los scripts Web, la descarga de aplicaciones de Internet o los correos con binarios adjuntos, ha provocado que el modelo tradicional de seguridad basado en cuentas de usuario haya dejado, en parte, de tener sentido, pues asume que todo el código, ya sea móvil o no, tiene el mismo nivel de confianza. Así, la plataforma .NET proporciona un modelo de seguridad basado en la evidencia, que posee un modelo de control de gran granularidad, pudiendo basarse o no en quien escribió el código, que intenta hacer dicho código, donde está instalado, y quien está intentando ejecutar dicho código.

Con estos objetivos, Microsoft .NET es una plataforma para construir, ejecutar y experimentar la tercera generación de aplicaciones distribuidas, que consiste en los siguientes elementos:

- Un modelo de programación basado en XML(*Extensible Markup Language*).

- Un conjunto de servicios Web XML para facilitar a los desarrolladores integrar estos servicios.
- Un conjunto de servidores que permiten ejecutar estos servicios
- Software en el cliente para poder utilizar estos servicios (como Windows XP, agendas electrónicas, etc.)
- Herramientas para el desarrollo

Figura 1.1: elementos de Microsoft .NET

.Net se encuentra dentro de un entorno en el cual hay muchos más productos y aplicaciones, y que en este caso, a diferencia de casi todos sus productos anteriores, Microsoft ha abierto hasta cierto punto su entorno, de forma que todo el mundo pueda participar en él.

El entorno dentro del que se encuadra .Net es una Internet que está cambiando de ser centrada en las personas, y basada en los contenidos, a estar centrada en las aplicaciones, y basada en los servicios. Estas aplicaciones y servicios forman parte de lo que se llaman servicios Web.

1.2 MICROSOFT .NET FRAMEWORK

El Microsoft .NET Framework es un importante componente en la familia de sistemas operativos de Microsoft Windows. Es la infraestructura de la plataforma Microsoft .NET. Es un ambiente común para la siguiente generación de aplicaciones que son fáciles de construir, desarrollar e integrar con otros sistemas en una red.

1.2.1 Componentes Básicos del .NET Framework

El .NET Framework consiste en dos partes principales: El Common Language Runtime (CLR) y la librería de clases de .NET Framework.

1.2.1.1 Common Language Runtime (CLR)

El motor de ejecución del CLR (*Common Language Runtime*) es el responsable de asegurar que el código es ejecutado como requiere, proporcionando una serie de facilidades para el código CIL (*Common Intermediate Language*) como:

- Carga del código y verificación.
- Gestión de las excepciones.
- Compilación "*Just In Time*" (JIT).
- Gestión de la memoria.
- Seguridad.

Lenguaje intermedio CIL(Common Intermediate Language)

El código intermedio CIL generado por los compiladores del framework .NET es independiente del juego de instrucciones de una CPU específica, pudiendo ser convertido a código nativo de forma eficiente. El lenguaje CIL es un lenguaje de un nivel de abstracción mucho mayor que el de la mayoría de los lenguajes máquina de las CPUs existentes, incluyendo instrucciones para

trabajar directamente con objetos (crearlos, destruirlos, inicializarlos, llamar a métodos virtuales, etc.), instrucciones para el manejo de excepciones, de tablas, etc.

La principal ventaja del CIL es que proporciona una capa de abstracción del hardware, lo que facilita la ejecución multiplataforma y la integración entre lenguajes. Otra ventaja que se deriva del uso de este lenguaje intermedio es la cuestión de la seguridad relativa a la verificación del código, pues el motor de ejecución puede examinar la intención del código independientemente del lenguaje de alto nivel utilizado para generarlo. Sin embargo, dado que las CPUs no pueden ejecutar directamente CIL, es necesario convertirlo a código nativo de la CPU antes de ejecutarlo.

Compilación JIT

La traducción de CIL a código nativo de la CPU es realizada por un compilador "*Just In Time*" o *jitter*, que va convirtiendo dinámicamente el código CIL a ejecutar en código nativo según sea necesario.

1.2.1.2 Librería de clases del .NET Framework

La librería de clases incluye un conjunto de paquetes de gran funcionalidad que los desarrolladores pueden usar para extender más rápidamente las capabilidades de su propio software. La librería incluye tres componentes claves:

- ASP.NET para ayudar a construir aplicaciones Web y servicios Web
- Windows Forms para facilitar el desarrollo de interfaces de usuario para clientes inteligentes
- ADO.NET para ayudar a conectar a las aplicaciones con las bases de datos.

1.3 COMPONENTES DE MICROSOFT .NET

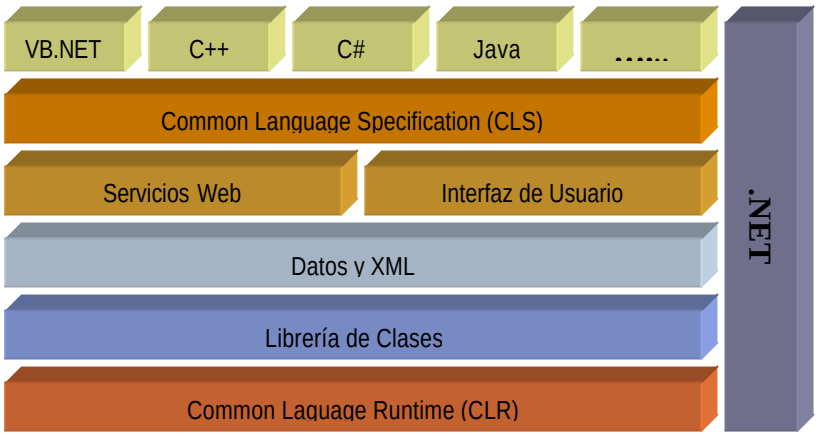


Figura 1.2: Componentes de .NET

El componente principal de .Net, que está en la capa más baja de su modelo de capas, es el CLR (*Common Language Runtime*), o máquina virtual común. Se trata de un programa, que se puede ejecutar, en principio, en cualquier sistema operativo, y que provee de una serie de servicios que se pueden usar desde diferentes lenguajes de programación.

Hay implementaciones no basadas en el código de Microsoft; la principal es el proyecto Mono, de la empresa Ximian[2]. La especificación de este CLR se quiere convertir en un estándar ECMA (*European Computer Manufactures Association*), de forma que pueda haber diferentes implementaciones de la misma, y diferentes lenguajes basados en ella.

Los ejecutables CLR (*Common Language Runtime*), están escritos en un lenguaje denominado MSIL (*Microsoft Intermediate Language*), similar al Java bytecode; en principio, cualquier programa escrito en MSIL (aunque nadie escribe en MSIL, se supone que lo hacen los compiladores) puede ejecutarse en cualquier sistema operativo donde funcione un CLR; el formato de esos

[2] Ximian® es lider en el proveer soluciones para servidor y escritorio en Linux

ficheros se denomina PE (*Portable Executable*). Además, el fichero ejecutable contiene metadatos, que informan sobre las funciones y tipos que implementan. Pero el concepto de ejecutable va un poco más allá: en .NET se usan ensamblajes, que pueden incluir partes de código, datos, códigos de seguridad, y todo lo necesario para convertirlo en código móvil y fiable (en el sentido de que esté firmado por alguien), que se pueda mover por Internet.

Los ensamblajes, a su vez, contienen metadatos, igual que sucede en los *.jar* de Java. Estos ejecutables y ensamblajes pueden ser generados a partir de diferentes lenguajes de alto nivel, pero los lenguajes deben de incluir dos cosas: un sistema común de tipos (CTS, *Common Type System*) y un sistema común de lenguajes (CLS, *Common Language Specification*).

El sistema común de tipos indica los tipos que tiene que soportar el lenguaje: tipos valor (por ejemplo, un entero; una variable entera contiene un valor) y tipos referencia, que apuntan a estructuras de datos dinámicas. Sin embargo, a diferencia de los lenguajes habituales, donde el tipo fundamental es un tipo valor, y las referencias son accesorias, y deben desreferenciarse para trabajar con ellas, en .NET el tipo fundamental es un objeto, y, de hecho, cualquier tipo valor se puede convertir en una referencia "encajándolo" (boxing).

El CTS (*Common Type System*) añade soporte para una serie de tipos que no se suelen encontrar en otros lenguajes: eventos (métodos que responden a un suceso determinado), propiedades (métodos para establecer y recuperar valores de variables de instancia) e indexadores, similares a los iteradores usados en otros lenguajes.

En cuanto a la especificación común de lenguaje (CLS, *Common Language Specification*), son una serie de reglas básicas requeridas para

integración del lenguaje, que garantice que el código intermedio generado desde cada uno de ellos sea interoperable con los otros. Hasta ahora, hay una serie de lenguajes propios de Microsoft: J# (similar al Java), VB.Net, Perl.Net, Python.Net. El más popular probablemente es C#, al que efectivamente, se le parece, pero que es un lenguaje totalmente diferente, con bastantes cosas originales. Todos los lenguajes usan el mismo conjunto básico de servicios, proporcionados por el CLR (*Common Language Runtime*): entrada salida, acceso al sistema de archivos, acceso a servicios remotos y acceso a datos.

El XML (*Extensible Markup Language*) está totalmente integrado con el C#: un programa permite pasar la definición de una clase a un fichero XML en formato XSchema (un formato que permite especificar, a su vez, el formato en el que se tiene que escribir un documento).

Hay una serie de problemas en este entorno: la disponibilidad del código fuente, ya que el MSIL (*Microsoft Intermediate Language*) se puede desensamblar con relativa facilidad, y además, tiene los metadatos que complementan más todavía la legibilidad del código; por eso, en caso de que no se quiera publicar el código, hay que hacer uso de algún tipo de técnica de enmascaramiento.

El siguiente problema son las prestaciones, problema común a todo tipo de máquina virtual; sin embargo, con el compilador JIT (*Just in Time*, similar al que tienen las máquinas virtuales Java), se trata de obtener el máximo rendimiento del código, compilándolo sólo cuando se le cargue por primera vez. Y, por supuesto, sobre todo esto está la sombra de la historia pasada de Microsoft: tener un estándar cerrado, que puede ser cambiado arbitrariamente, lo cual puede dejar fuera del negocio a muchos.

Otro problema adicional es la falta de servicios de autentificación. Inicialmente, se iba a usar *Passport*, que luego se convirtió en *Hailstorm*, para acabar siendo *My Services*. Finalmente, por falta de apoyo por parte de la industria, Microsoft decidió suprimirlo. Nadie quería, como es natural, que fuera Microsoft quien autentificara a sus clientes, por mucho que sea Microsoft.

Otro posible problema son los virus; como cualquier formato ejecutable, el PE (*Portable Executable*) se puede infectar con virus, y si el CLR (*Common Language Runtime*) se convierte en ubicuo, puede tener bastantes posibilidades de propagación.

Los demás componentes de .Net permiten extender a todos los productos de Microsoft la funcionalidad de .Net:

- ASP.NET: *Active Server Pages*, en su versión para .Net.
- VB.NET: versión para el CLR del Visual Basic, el lenguaje común a todas las aplicaciones de Microsoft.
- ADO.NET, acceso a objetos de datos (*Access to Data Objects*), que permite acceder de forma orientada a objetos a bases de datos; también da una serie de servicios para acceso a bases de datos y otros repositorios de objetos desde dentro de la CLR.
- Perl.NET, Python.NET son desarrollos de *ActiveState*, que se integran con el entorno Visual Studio.NET y permiten desarrollar programas en esos lenguajes.
- WinForms, diseño gráfico de ventanas dentro de .NET. En la implementación Mono, se sustituye por Gtk#.

1.4 USANDO .NET

Hay dos vías principales: la vía Microsoft y la vía del código abierto. Ambas son gratuitas; como el entorno está en sus principios, todas las herramientas, por el momento, son gratuitas.

La vía Microsoft incluye bajarse el .Net framework SDK (*Software Development Kit*) junto con el primer Service Pack. Estos paquetes incluyen todo lo necesario para desarrollar aplicaciones para .NET: entorno, CLR, ASP.NET. Para usarlo, es necesario tener un Windows de la familia NT: WNT, W2000 o WXP; no funciona sobre Windows 9X.

Un producto comercial, Visual Studio .NET, sirve también para desarrollar .Net en un entorno mucho más amigable, y hace mucho más fácil usar los formularios que son parte del entorno, WinForms y WebForms. Recientemente, Microsoft también ha sacado, como herramienta gratuita, Web Matrix para desarrollar ASP.NET de forma visual.

La otra vía incluye varios proyectos libres y gratuitos. El proyecto más importante es Mono, una implementación del compilador de C#, del CLR y de la librería básica de clases de C#. La licencia de C# es libre, al igual que la documentación. Otra alternativa, aunque mucho menos desarrollada, es dotGNU, que trata de crear una plataforma de servicios Web libres (por lo pronto se llama portable.Net) y un sistema de autentificación (que se denominan Identidades Virtuales), pero no se encuentra tan desarrollado como Mono.

1.5 COMPETIDORES Y FUTURO DE .NET

Como principal competidor se presenta J2EE (*Java 2 Enterprise Edition*), una versión de Java con librerías de clase añadidas, que usa la

máquina virtual Java, y tiene muchas características similares a .Net. Java es un lenguaje bastante maduro, con soporte de cientos de librerías fuera de las básicas, y con una comunidad bastante extensa. En ese sentido, C# vs. J2EE puede tratarse de una batalla de "comunidad" frente a Microsoft, y no está claro quién la va a ganar.

Lo que sí está claro es que Microsoft apuesta por .Net, como centro de su estrategia, y que cuando Microsoft apuesta por algo, acaba ganando. Es posible que coexistan las dos plataformas, y es posible que se abran la una a la otra; por ejemplo, que haya intérpretes CLR (*Common Language Runtime*) que corran dentro de una JVM (*Java Virtual Machine*) o viceversa.

La apuesta que no se puede perder es la apuesta por los servicios Web, y aplicaciones basadas en XML. Todos los grandes de la industria apuestan por ellas, y gran parte de las aplicaciones de cara al usuario, el middleware y los servidores entenderán y servirán XML. Es decir, que independientemente de la plataforma, XML será el vencedor.

1.6 COMMON LANGUAGE INFRASTRUCTURE (CLI)

Es un estándar ECMA (ECMA-335) que permite a las aplicaciones que sean escritas en una variedad de lenguajes de programación de alto nivel y ejecutadas en diferentes sistemas operativos.

Los lenguajes de programación que integran el CLI (*Common Language Infrastructure*) tienen acceso a las mismas librerías de clases básica y son capaces de ser compiladas en el mismo lenguaje intermedio (IL – *Intermediate Language*).

El CLI (*Common Language Infrastructure*) provee una especificación para el código ejecutable y para el ambiente de ejecución Virtual Execution System (VES) en el cual corre.

En el centro del CLI (*Common Language Infraestructure*) está un sistema de tipos unificado, el Common Type System (CTS), el cual es compartido por compiladores, herramientas, y el propio CLI. Este es el modelo que define las reglas que el CLI sigue cuando se declara, usa y maneja tipos. EL CTS (*Common Type System*) establece un armazón que permite la integración entre lenguajes, seguridad de tipos, y alto rendimiento en la ejecución del código.

Forman parte de esta infraestructura:

- Common Type System (CTS), el cual provee un rico sistema de tipos que soportan todas las operaciones encontradas en muchos lenguajes de programación. Este sistema trata de soportar la completa implementación de un amplio rango de lenguajes de programación.
- Metadatos. El CLI usa metadatos para describir y referenciar tipos definidos por el CTS (*Common Type System*). Los metadatos están almacenados de una forma que es independiente de cualquier lenguaje de programación particular. Entonces, los metadatos proveen un mecanismo de intercambio común entre herramientas que manipulan programas (compiladores, depuradores, etc.)
- Common Language Specification (CLS). Es un acuerdo entre los diseñadores de lenguajes y los diseñadores de librerías de clase. Especifica un conjunto de convenciones del CTS (*Common Type System*), de manera que los lenguajes provean a los usuarios la gran habilidad de acceder a las librerías de clase.
- Virtual Execution System (VES). Implementa y hace cumplir el modelo CTS (*Common Type System*). Es el responsable de la carga y corrida

de programas escritos por el CLI (*Common Language Infraestructure*). Provee los servicios necesarios para ejecutar código manejado y datos, usando los metadatos para conectar separadamente los módulos generados conjuntamente en tiempo de ejecución.

Todos estos aspectos forman una estructura para diseño, construcción, desarrollo y ejecución de componentes y aplicaciones. El conjunto apropiado del CTS es utilizado por cada lenguaje de programación que aplica el CLI. Las herramientas basadas en los lenguajes se comunican entre si y con el VES (*Virtual Execution System*) usan los metadatos para definir y referenciar los tipos usados para construir una aplicación. El *VES* usa los metadatos para crear instancias de los tipos de acuerdo a sus necesidades y para proveer información de los tipos de datos y otras partes de la infraestructura como son servicios remotos, seguridad, etc.

1.6.1 Common Type System (CTS)

Para conseguir la interoperabilidad entre lenguajes es necesario adoptar un sistema de tipos común. Así, el sistema de tipos común (CTS, *Common Type System*) define como se declaran, utilizan y gestionan los tipos en el CLR (*Common Language Runtime*).

El CTS desarrolla las siguientes funciones:

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

1.6.1.1 Clasificación de tipos

El CTS se divide en dos categorías generales de tipos:

- Tipos Valor. Las instancias de los tipos Valor son almacenadas como la representación de su valor como una secuencia de bits en memoria, careciendo del concepto de identidad. Dentro de los tipos valor se encuentran los predefinidos (implementados por el CLR), los definidos a medida por el usuario, y las enumerados.
- Tipos Referencia. Las instancias de los tipos Referencia son almacenadas como referencias a la localización de su valor. Los tipos referencia son una combinación de una localización, su identidad, y una secuencia de bits (su valor). Dentro de los tipos referencia se encuentran los tipos interfaz, los tipos punteros, y los tipos autodescriptivos. Los tipos autodescriptivos son aquellos en los cuales es posible obtener el tipo de su valor por inspección; todos los tipos autodescriptivos heredan de la clase base *Object*, encontrándose dentro de esta categoría los arrays y las clases, dividiéndose esta última en clases definidas por el usuario, tipos Valor encajados (boxed), y los delegados.

En la terminología del framework .NET, una instancia de un *tipo Valor* o de un *tipo Referencia* es conocida como *valor*, de forma que el valor de un *tipo Valor* es su representación binaria, mientras que el valor de un *tipo Referencia* es la localización de la representación binaria. Cada valor tiene un tipo, el cual define su representación y las operaciones que pueden ser invocadas sobre dicho tipo; sin embargo, en tiempo de ejecución no siempre es posible determinar el tipo de un valor por inspección, como es el caso de los *tipos Valor* y los *tipos Punteros*.

1.6.1.2 Definición de tipos

Una definición de un tipo construye un nuevo tipo a partir de tipos existentes. Los *tipos valor* predefinidos, los punteros, arrays y delegados son definidos al ser utilizados, por lo que a estos tipos se les conoce como tipos implícitos. La definición de un tipo incluye los siguientes elementos:

- Los atributos definidos sobre el tipo (como se verá en la sección de los metadatos, los atributos son un mecanismo de extensión de los mismos).
- La visibilidad del tipo. Un tipo puede ser visible a todos los ensamblados (visibilidad pública), o sólo para el ensamblado que lo define (visibilidad de ensamblado).
- Nombre del tipo. Un tipo queda definido dentro de un ensamblado, por lo que sólo tiene que ser único dentro del ensamblado.
- El tipo base del tipo definido. Un tipo definido sólo puede tener un tipo base.
- Las interfaces implementadas por el tipo.
- Las definiciones de cada uno de los miembros del tipo. Dentro de un tipo pueden definirse los siguientes miembros:
    - Eventos. Definen incidentes a los que se puede responder, así como los métodos para suscribirse / de suscribirse de recibir la notificación del evento, y métodos para la generación del evento. Los eventos son implementados mediante delegados.
    - Campos (variables). Describen y contienen el valor de un tipo.
    - Tipos anidados. Definen a un tipo dentro del ámbito del tipo que lo contiene.
    - Métodos. Definen las operaciones disponibles para un tipo.
    - Propiedades. Nombran a un valor lógico o al estado de un tipo, y constituyen una alternativa a los tradicionales métodos de acceso o modificación. Las propiedades pueden contener lógica interna,

así como lanzar excepciones si fuera necesario. La utilización de las propiedades es idéntica a la de los campos, es decir, utilizando la notación punto *Objeto.Propiedad*. Las propiedades son utilizadas frecuentemente para mantener la interfaz pública de un tipo independiente de la representación actual de dicho tipo.

1.6.2 Common Language Specification (CLS)

El CLR (*Common Language Runtime*) proporciona, mediante el sistema de tipos común CTS (*Common Type System*) y los metadatos, la infraestructura necesaria para lograr la interoperabilidad entre lenguajes, pues todos los lenguajes siguen las reglas definidas en el CTS para la definición y el uso de los tipos, y los metadatos definen un mecanismo uniforme para el almacenamiento y recuperación de la información sobre dichos tipos. Pero a pesar de esto, no hay ninguna garantía de que la funcionalidad de los tipos escritos por un desarrollador en un lenguaje determinado pueda ser completamente utilizado por otros desarrolladores que utilizan otros lenguajes, pues cada lenguaje de programación utiliza los elementos del CTS y de los metadatos que necesita para soportar su propio conjunto de características, pudiendo existir características del CLR que no son soportadas por un lenguaje concreto.

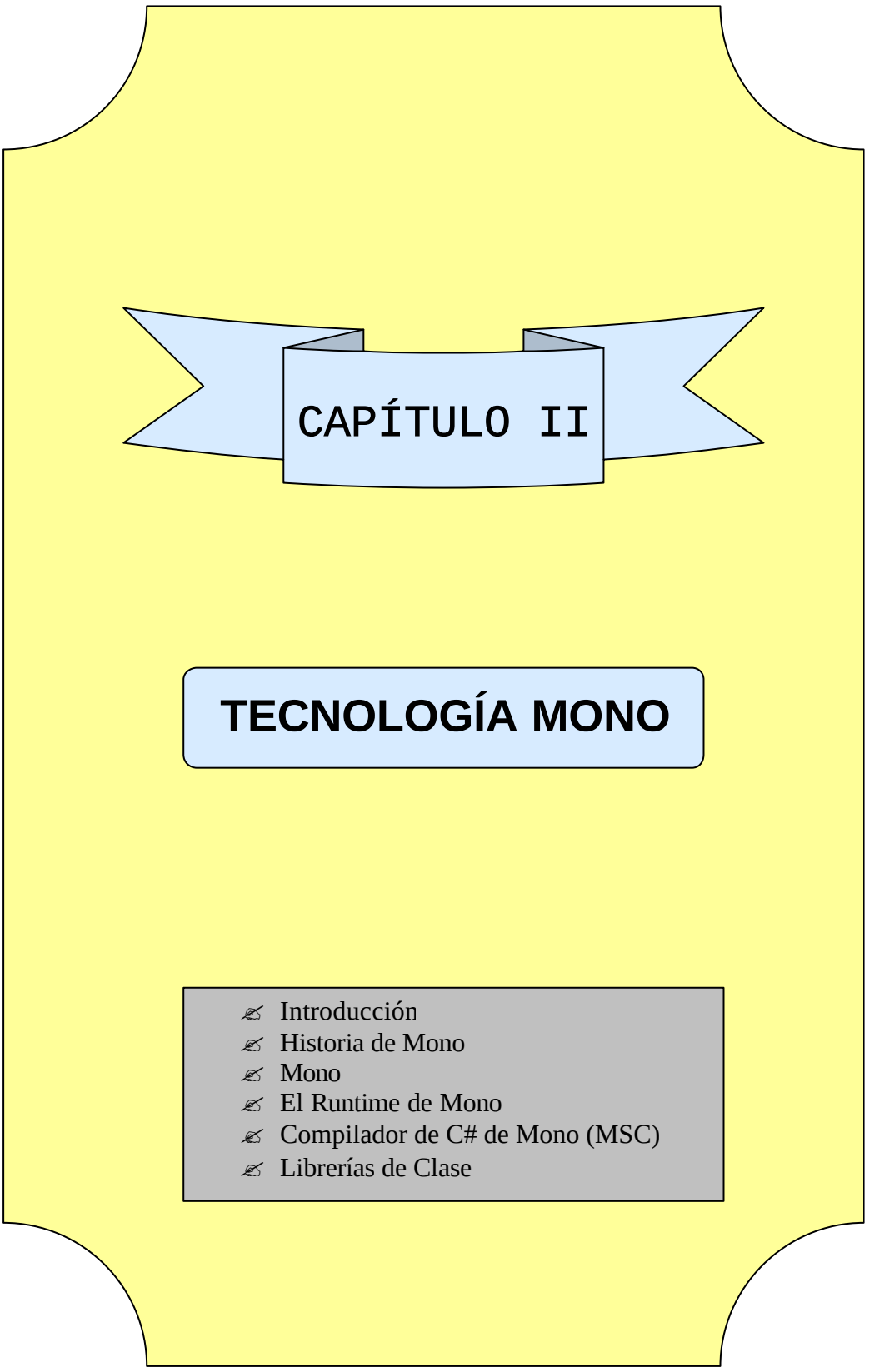
Para asegurar que el código escrito en un lenguaje sea accesible desde otros lenguajes se ha definido la Especificación del Lenguaje Común (CLS, *Common Language Specification*), que establece el conjunto mínimo de características que deben soportarse para asegurar la interoperabilidad, siendo dicho conjunto de características mínimas un subconjunto del CTS. El CLS ha sido diseñado para ser lo suficientemente grande como para que incluya las construcciones que son utilizadas comúnmente en los lenguajes, y lo suficientemente pequeño para que la mayoría de los lenguajes puedan cumplirlo.

Así, para que un objeto pueda interactuar con otros objetos, independientemente del lenguaje en el que hayan sido implementados, estos objetos deben exponer únicamente las características que están incluidas en el CLS. Los componentes que están adheridos a las reglas del CLS y utilizan sólo las características incluidas en el CLS se denominan como componentes conformes con CLS (*CLS-compliant*). Sin embargo, el concepto de conforme con CLS tiene un significado más específico dependiendo de si trata de código conforme con CLS o de herramientas de desarrollo conformes con CLS.

Bibliografía

Referencias WWW

- http://www.computerhope.com/msdos.htm
- http://www.cetus-links.org/oo_distributed_objects.html
- http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto-corba.html
- http://www.ecma.ch
- http://www.ximian.com
- http://www.microsoft.com/net/basics/whatis.asp
- http://www.w3schools.com/ngws/default.asp



TECNOLOGÍA MONO

- Introducción
- Historia de Mono
- Mono
- El Runtime de Mono
- Compilador de C# de Mono (MSC)
- Librerías de Clase

2.1 INTRODUCCIÓN

A finales del 2000 Microsoft publica los primeros documentos sobre la tecnología .NET. En estos se especificaba el funcionamiento de esta nueva plataforma que nacía entre otros motivos para hacer frente al éxito de **Java** de la competidora **Sun**.

La idea de .NET tiene bastantes similitudes con la tecnología Java, ambos compilan el código fuente a un código intermedio (no directamente a código máquina). En el caso de Java este código es llamado *bytecode* y en .NET recibe el nombre de CIL (*Common Intermediate Language*).

Para ejecutar este código intermedio es necesario un entorno que lo interprete y así poder pasar al código máquina correspondiente al sistema/arquitectura donde se este ejecutando. De esta forma se consigue independencia del ejecutable en contraposición al tradicional compilado a código máquina, ya que este último solo podría ser utilizado en máquinas que soporten el mismo conjunto de instrucciones y en sistemas que conozcan el formato de ese ejecutable.

Pero .NET va más allá, su objetivo no es sólo la independencia del compilado sino también la independencia del lenguaje de alto nivel, es decir, CIL ha sido especialmente diseñado para proporcionar todo lo necesario a la mayoría de lenguajes actuales. El lenguaje que aprovecha toda la potencia de CIL es C# diseñado por la propia Microsoft, pero esto no impide que todo aquel que quiera formar parte de la plataforma .NET construya un compilador de su lenguaje a código intermedio CIL.

Esto nos proporciona por ejemplo la posibilidad de poder reutilizar clases programadas en lenguaje C# desde ***Visual Basic.NET*** de forma muy sencilla, cosa que hasta el momento sólo era posible mediante complejos

mecanismos poco flexibles y que ahora es posible tener de forma nativa a la plataforma.

.NET tiene definido un CTS (*Common Type System*) con los tipos de datos soportados, los cuales son suficientes para cubrir cualquier lenguaje actual. Pero esto no es suficiente para garantizar la interoperabilidad entre lenguajes, porque por ejemplo imaginemos que hay un lenguaje que soporta el tipo entero sin signo y otro que no, esto impediría la interoperabilidad entre ambos lenguajes. Por ese motivo también se ha definido el CLS (*Common Language Specification*), el cual es necesario que cumplan todos los lenguajes que quieran poder disfrutar de dicha interoperabilidad.

Esta es la gran diferencia básica con respecto a *Java* de *Sun*, ya que el bytecode no ha sido diseñado para tales practicas (compilación de cualquier lenguaje actual a bytecode) y por tanto no resulta óptima para desempeñar las tareas que puede cubrir CIL.

Las ventajas globales que tenemos con .NET son muy importantes:

- Independencia del sistema/arquitectura: En todo sistema/arquitectura donde este implementado un intérprete de código CIL se podrá ejecutar nuestro programa.
- Independencia del lenguaje: Seria posible mezclar grupos de programadores expertos en diferentes lenguajes.
- Reutilización de código: Por ejemplo, seria posible utilizar código antiguo implementado en lenguaje *Visual Basic.NET* en otro proyecto actual que este trabajando con el lenguaje C#.

La plataforma .NET esta compuesta por dos pilares fundamentales:

- Common Language Runtime (CLR)

Este es el entorno de ejecución que traducirá el código intermedio CIL a código máquina y por tanto permitirá ejecutar cualquier aplicación de la plataforma.

Algunas implantaciones del CLR tienen incorporado el JIT (*Just in Time*) de forma que sólo se traduce a código máquina las partes necesarias y se recuerdan por si vuelven a ser llamadas (ejemplo: las funciones). Así se consigue un mayor rendimiento de ejecución.

- Framework Class Library (FCL)

Librería de clases que proporciona una gran cantidad de servicios: Entrada/Salida, XML, ADO.NET (acceso a Bases de datos), Windows.Forms (aplicaciones gráficas), sockets, colecciones, hilos, etc.

La FCL presta sus servicios a cualquier lenguaje que este dentro de la plataforma .NET ya que esta implementada en lenguajes que cumplen la CLS (en el caso de mono, esta implementada íntegramente con C#), minimizando así las características propias de cada lenguaje.

Hasta ahora se he dado una visión global de los beneficios de la tecnología .NET pero lo realmente interesante es que Microsoft ha estandarizado a través del organismo ECMA los elementos más importantes:

- Common Language Infrastructure (CLI) lo que incluye:
  - Common Type System (CTS)
  - Common Language Specification (CLS)
  - Virtual Execution System (VES)
  - Metadata Definitions and Semantics
  - Common Intermediate Language (CIL)
- C# Language Specification

2.2 HISTORIA DE MONO

En Diciembre del 2000 Miguel de Icaza (Co-fundador de la empresa *Ximian*, fundador y presidente de la *GNOME Foundation*) se interesó bastante por la tecnología .NET al tener acceso a los primeros documentos de Microsoft.

GNOME (GNU Network Object Model Environment) siempre había luchado por proporcionar facilidades al programador y una de las características más conocidas es que existen multitud de *bindings* (adaptadores) para poder utilizar cualquier lenguaje para desarrollar aplicaciones. Pero la elaboración de dichos *bindings* era tremendamente laboriosa y cada vez que se realizaba un cambio en la interfaz original, era necesario cambiar todos y cada uno de los *bindings*.

Para intentar mejorar y facilitar la reutilización de código se realizó una implementación de componentes utilizando *CORBA* llamada *Bonobo*. Pero tampoco ha tenido éxito ya que era necesario que todo el mundo utilizase esa característica y eso no fue así.

Por tanto, con .NET se abrió una nueva puerta para conseguir hacer de *GNOME* en un futuro, un escritorio mejor y más atractivo tanto para usuarios como para programadores. Con esta tecnología por fin se consiguió lo que el proyecto *GNOME* siempre había buscado, independencia del lenguaje para programar en dicho escritorio.

Además de estos beneficios, la mayor parte de .NET estaba estandarizado y por tanto era viable la implementación de una plataforma libre que incorporase compiladores, runtimes y librerías de clases. Pero Miguel tenía dos grandes obstáculos con .Net: no era gratuito y no trabajaba sobre

Linux. Con estos problemas, de Icaza y su equipo empezó la construcción de su propia versión de .Net.

Cuando Mono lanzó la nueva versión de su software, basado en C# y en los componentes estándar CLI (*Common Language Infrastructure*) de .Net y añadidas a los estándares ECMA (*European Computer Manufacturers Association*) e ISO (*International Organization for Standardization*), Microsoft publicó y abrió los API's para .Net. Este fue el mayor paso de Microsoft desde que se lanzó la tecnología, y abrió la puerta a otros para la construcción de implementaciones compatibles del API .Net.

Y la gente empezó a cruzar por esa puerta. El proyecto Mono, liderado por Miguel de Icaza, y Microsoft .Net empezaron a diseñar una implementación de la plataforma de desarrollo .Net. Ambos, .Net y Mono proveen los mismos API's, los cuales pueden ser llamados desde múltiples lenguajes. Ellos también proveen integración simplificada con los lenguajes, y el CLR (*Common Language Runtime*), el cual es similar al JVM (*Java Virtual Machine*) de Java.

A de Icaza no le importó que el y su equipo hayan seguido el liderazgo de Microsoft. Y efectivamente mucha gente estuvo y está todavía interesada en .Net como tecnología, y no les importa que esta tecnología haya sido inventada por Microsoft. Cualquier tipo de problemas se resuelven directamente por .Net, dijo. Y GNOME era un buen lugar para lo que .Net estaba tratando de resolver. El equipo del proyecto Mono buscaba soluciones y encontraron en .Net lo necesario para ello.

La respuesta de Microsoft al código abierto fue menos delicada y más combativa. Pero últimamente los mira como la relación que se esta envolviendo en una balance productivo.

Icaza hizo un comentario en el cual Microsoft planeaba compartir la propiedad intelectual relacionada a .Net. En espera de la revisión presentada por los abogados de Microsoft, dijo: “Microsoft patenta sobre el desarrollo de la tecnología, específicamente para .Net; y estará de acuerdo con aquellos que traten de implementarlo”

Esta fue una gran noticia, desde que hubo una discusión en sitios Web de código abierto, en los cuales se comentaba que patentar y poner una propiedad intelectual a este asunto podría ser usado por Microsoft para frustrar cualquier implementación no .Net, a pesar de la presentación de .Net como un estándar.

Microsoft fue más cauteloso en este asunto. John Montgomery, director de la Administración de Producto de Microsoft, dijo que Microsoft seguirá todas las políticas de estandarización ECMA e ISO. Esto especificaba algo “razonable y no discriminatorio” a través de una política de patente no necesariamente libre. Cuando se habló de “razonable y no discriminatorio”, se sabía que no mataría al proyecto Mono, simplemente tendría un efecto desalentador fuera del mundo Windows.

En un nivel de ingeniería, la relación entre el equipo del proyecto Mono y la gente de Microsoft fue amigable. Se encontraban en reuniones del ECMA y otros eventos. Era una simpatía natural, debido a que ambos trabajan en cosas similares, las mismas herramientas, y ambos interesados en lo que sucedía con el mundo .Net.

Montgomery tuvo afectos similares. Sostuvo que el hecho de que Ximian como empresa estaba haciendo este trabajo era muy gratificante. Una validación del trabajo que había hecho la gente de Microsoft.

Un efecto de todo esto fue, que los desarrolladores que no habían considerado a .Net, estaban próximos a probarlo, pues conocer sus productos tenía un potencial al desarrollo en más de una plataforma. Esto no necesariamente significaba perdidas en las ventas para Microsoft. En efecto, esto expandía la posibilidad a la oportunidad de Microsoft de vender Visual Studio .Net a un nuevo grupo de desarrolladores, también iba a hacer posible el tener un trabajo en el desarrollo utilizando Visual Studio .Net, y después ponerlo a producir en un servidor .Net o en Mono en Linux, como un cliente o como la situación lo demande.

2.3 MONO

Mono es la plataforma de desarrollo creada por la empresa Ximian, y que se basa en Microsoft .Net para el desarrollo, despliegue y ejecución de aplicaciones orientadas a servicios sobre entornos altamente distribuidos, tal y como lo es Internet. Es un proyecto que mejora considerablemente el desarrollo, despliegue y ejecución de las aplicaciones, e introduce el concepto de los servicios Web, que permiten el desarrollo de aplicaciones hábilmente acopladas basadas en componentes que utilizan protocolos de comunicación estándares de Internet como SOAP(*Simple Object Access Protocol*) y XML(*Extensible Markup Language*).

Mono es una implementación de código abierto del Microsoft .Net Framework, el cual incluye: un compilador para el lenguaje C#, un motor runtime compatible con el estándar ECMA(*European Computer Manufacturers Association*) para el Lenguaje Común en Tiempo de Ejecución (CLR, *Common Language Runtime*) y un conjunto de librerías de clase, sobre la cual se sitúa la parte relativa a las interfaces de usuario, que son los **Windows Forms** para las aplicaciones de ventana y **ASP.Net** para las aplicaciones Web. A continuación se muestra un esquema que muestra todos estos componentes:

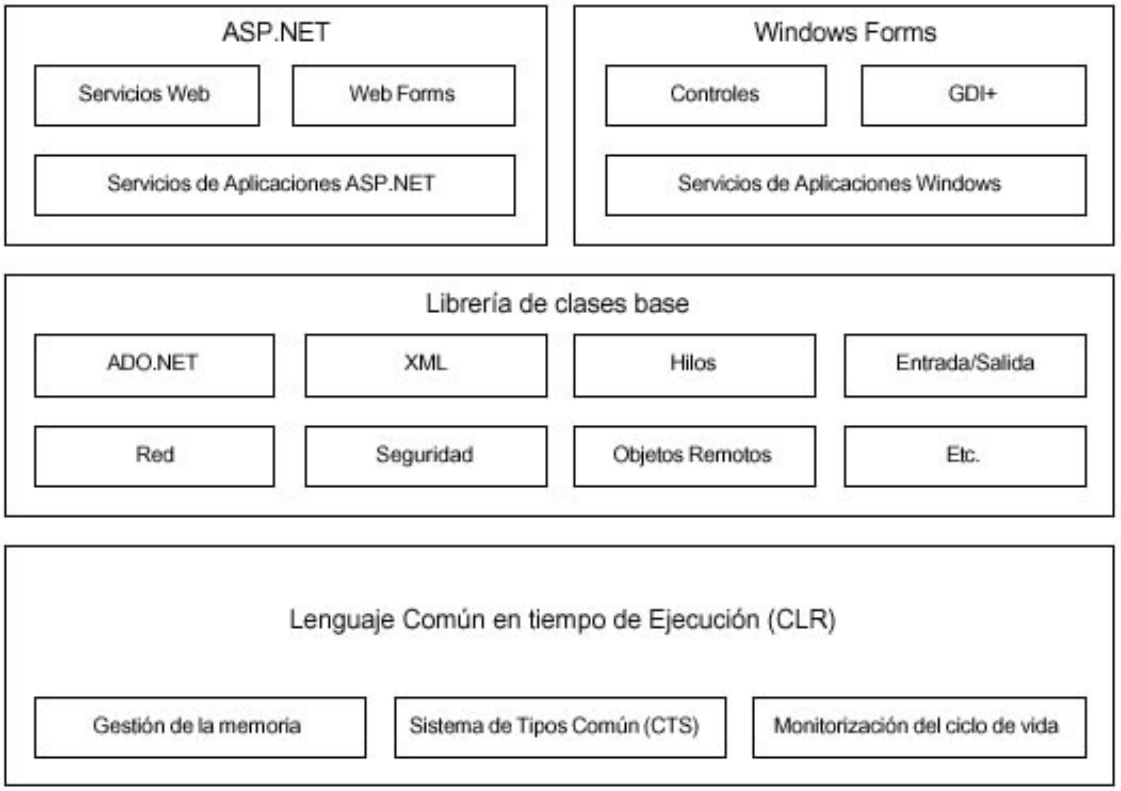


Figura 2.1: Componentes básicos de .Net

Microsoft creó esta nueva plataforma de desarrollo conocida como .Net, pero es muy importante lo que la comunidad de software de código abierto ha realizado creando el proyecto Mono como una implementación de código abierto de Microsoft .Net.

Mono implementa las siguientes partes de la tecnología .NET:

- Common Language Runtime
- Compilador/Desensamblador IL
- Compilador C# (MCS)
- Compilador Visual Basic.NET (MBAS)
- Librería de clases

Los compiladores proporcionados están bajo la licencia **GNU GPL**[3], el runtime tiene la licencia **GNU LGPL**[4] y la librería de clases tiene la licencia **MIT X11**[5].

Actualmente Mono se puede ejecutar en los siguientes sistemas:

Plataforma	Interprete	Jitter (Just in Time)
MS Windows	Sí	Sí
MacOSX	Sí	No
FreeBSD	Sí	Sí
GNU/Linux (x86)	Sí	Sí
GNU/Linux (PPC)	Sí	No
GNU/Linux (S390)	Sí	No

Actualmente el compilador de C# y el runtime se encuentran en un estado muy avanzado, ya se puede utilizar perfectamente para cualquier proyecto. El compilador de ***VisualBasic.NET*** aún necesita crecer. La librería de clases está en proceso aunque ya es suficientemente madura y útil como para poder ser utilizada exceptuando ***Windows.Forms*** (aplicaciones gráficas, aunque existe un proyecto en muy buen estado llamado ***GTK#***[6] con el cual podemos realizar aplicaciones utilizando las librerías gráficas ***GTK***) y ***EnterpriseServices***.

En cuanto a la implementación .NET de Microsoft solo cubre sus sistemas MS Windows además de *MacOSX* y *FreeBSD*. Ofrecen el código del Common Language Infrastructure, es decir, el Runtime y el compilador de C# entre algunas otras pequeñas utilidades pero no toda la implementación de la librería de clases. Este código es ofrecido bajo la licencia *Microsoft Shared*

[3] GNU GPL: información en: http://www.opensource.org/licenses/gpl-license.html
[4] GNU LGPL: información en: http://www.opensource.org/licenses/lgpl-license.html
[5] MIT X11: información en: http://www.opensource.org/licenses/mit-license.html
[6] GTK#::Información en: http://gtk-sharp.sourceforge.net

Source CLI, C#, y jscript License[7], la cual limita completamente al usuario impidiéndole realizar ninguna modificación y mucho menos redistribuirlo. Del acceso a dicho código podrían incluso derivarse problemas legales si se llegase a inspirar algún programador en el código mostrado.

En cambio Mono ofrece todo su código, compiladores, runtime, librería de clases; esto con licencias libres, perfectas para poder aprender sin preocuparse de posibles problemas legales futuros y con la posibilidad de realizar modificaciones en el código para colaborar o distribuirlo ya sea comercialmente o de forma gratuita.

Mono también pretende implementar completamente las partes de .NET que no están estandarizadas como *ADO.NET* (acceso a Bases de datos), *ASP.NET* (desarrollo de páginas Web) y *Windows.Forms* (aplicaciones gráficas). De esta forma también se puede conseguir una sencilla migración de la plataforma .NET implementada por Microsoft a Mono.

Esto podría llevar a plantearse un posible problema futuro, si Microsoft patentase alguna de esas partes o hiciese que su implementación .NET incumpliese el estándar, ¿podría perderse todo el trabajo realizado con Mono? En absoluto, el núcleo de .NET está estandarizado y por tanto libre de patentes. En caso de que Microsoft incumpliese el estándar en su implementación se originaría una situación donde el Framework .NET de MS y Mono serían incompatibles, pero Mono aún seguiría siendo útil por si mismo.

En el caso de que se hagan patentes sobre las partes no estandarizadas, Mono puede optar por varias soluciones:

[7] Licencias de Microsoft para CLI: http://msdn.microsoft.com/MSDN-FILES/027/002/097/ShSourceCLILicense.htm

1) Intentar evitar los problemas de la patente realizando una implementación diferente de la funcionalidad de forma que no se vea afectada la API.

2) Eliminar las partes afectadas por la patente.

3) Encontrar ejemplos de implementaciones anteriores que hagan inválida la patente. Por ejemplo, muchas de las funcionalidades de la librería de clases ya existían antes en Java y por tanto esto invalidaría una patente.

A todo esto también cabe destacar que aquellos países donde las patentes de software no sean válidas no se verían afectados por estas posibles acciones.

2.4 EL RUNTIME DE MONO

El runtime de Mono implementa un motor JIT (*Just In Time*) para la máquina virtual CIL (*Common Intermediate Language*), el cargador de clases, el recolector de basura, un sistema de gestión de memoria y librerías de acceso a metadatos.

Sus características son:

- Carga y verificación de las imágenes CIL (*Common Intermediate Language*)
- Ejecución de las imágenes CIL
- Recolección de basura
- Soporte multihilo y de E/S
- Interacción con librerías ya existentes (P/Invoke)
- Sistema de seguridad

Se tienen dos runtimes:

- **mono:** es el compilador JIT (*Just in Time*), el cual usa un selector de instrucción *BURS*[8].
- **mint:** Es un interprete de Mono. Es un motor runtime muy fácil de portar

Mint fue originalmente desarrollado como una prueba del concepto de mono. Fue diseñado para ser fácil de depurar, fácil de estudiar, y los suficientemente comprensivo como para ser usado como una referencia para los problemas de depurado con el motor JIT. Mint es más portable que JIT, y un efecto de esto es que se puede correr Mono en diferentes arquitecturas sin mucho trabajo. Idealmente, se ha creado JIT para cada plataforma soportada, pero el intérprete es más útil cuando se quiere correr a Mono muy rápido, y correrlo bajo sistemas donde la velocidad es lo más importante.

2.4.1 Mono JIT

El JIT (*Just in Time*) de Mono traslada las instrucciones CIL a código nativo en tiempo de ejecución. El JIT compila un ensamblado entero en una sola pasada, o un método a la vez; y en la primera pasada cada método es invocado individualmente.

El JIT usa un conjunto de macros que generan código en un buffer de memoria. Mono necesita un conjunto de macros por cada arquitectura. Estos macros simplifican la generación de código, el depurado y el prototipado. La Interfaz de generación de código para la plataforma de computadora de clase x86 se encuentra en el archivo *mono/arch/x86/x86-codegen.h*. Los macros *x86-codegen.h* se originaron en Plataforma de Maquina Virtual de Java de Intel. El equipo de Mono convirtió esos macros para ser usados desde C, el lenguaje en el cual esta escrito el JIT.

[8] BURS: Generador de Código. Referencias: http://www.research.microsoft.com/~drh/pubs/iburg.pdf

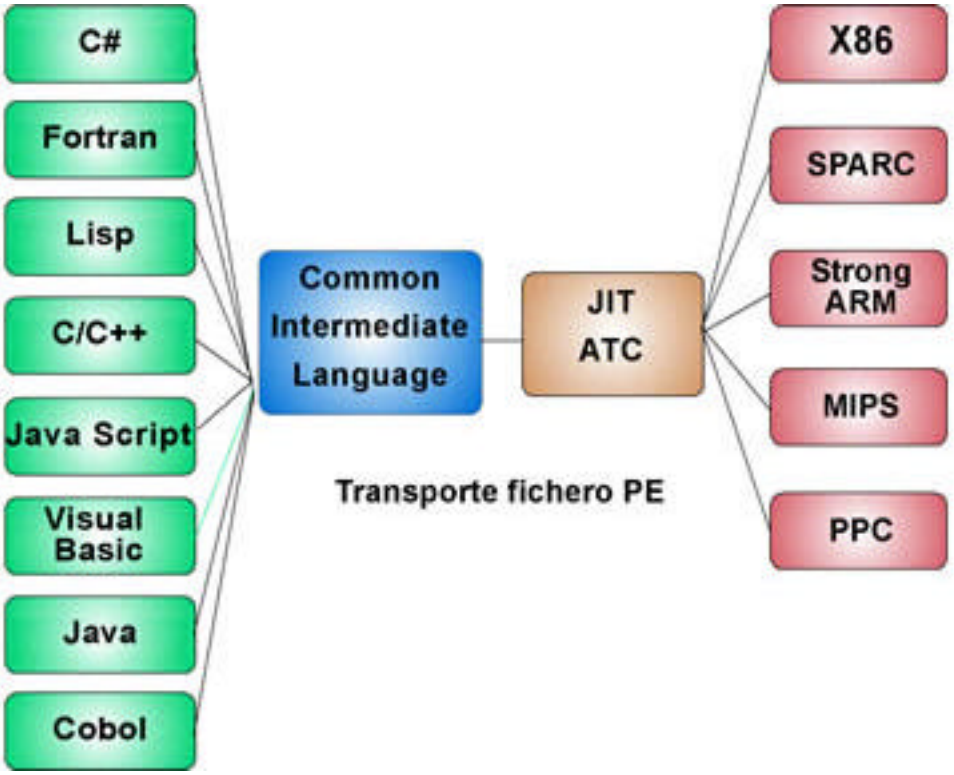


Figura 2.2: Compilación JIT en sus varios pasos

La conversión del código CIL a instrucciones nativas es donde las cosas se tornan interesantes. Mono usa un selector de instrucción basado en el sistema BURS (B*ottom-up Rewrite Systems*), la misma tecnología usada por el compilador portable *Icc ANSI C*.

BURS usa una gramática que mapea un conjunto de operaciones (los nodos terminales) a elementos no terminales que se combinan con la arquitectura destino. Esta gramática esta aplicada en un programa generador de código, *monoburg*[9]. Los conflictos son resueltos usando funciones asociadas con cada producción. La combinación de patrones de entrada es un árbol de operaciones. Este mapea el árbol a la arquitectura destino seleccionando los nodos que tienen un costo total mínimo asociado con ellos.

[9] monoburg: selector de instrucción BURS implementado en el proyecto Mono

El primer paso transforma una secuencia de instrucciones CIL en un bosque de árboles. Cada árbol tiene que ser alimentado al selector de instrucción separadamente. Durante este proceso de creación de bosques y árboles, las instrucciones CIL estándar son transformadas en códigos que son mejor combinadas con el selector de instrucción.

Para generar el código, un número de pasadas son realizadas en los árboles de nodos. El primer pase etiqueta todos los nodos y encuentra el árbol más económico, y en el segundo pase realiza la asignación del registro. La etapa final emite el código x86.

En esta escritura, el motor JIT (*Just in Time*) soporta la mayoría de las características no orientadas a objetos de la máquina virtual. Para cuando se las lea, las características orientadas a objetos deberían ser implementadas.

2.4.2 Recolección de Basura (Garbage Collection, GC)

Recolección de basura (GC[10]) es una parte del sistema en tiempo de ejecución de los lenguajes, o una librería añadida, puede ser asistida por el compilador, el hardware, el sistema operativo, o cualquier combinación de los tres, que automáticamente determina que parte de la memoria de un programa no se está usando, y lo recicla para otros usos. Esto también se lo conoce como “reclamación de almacenamiento (o memoria) automática”. Y esto es muy bueno, porque la administración manual de memoria es propensa a errores. La mayoría de programas todavía contienen filtraciones.

Un segundo beneficio de la recolección de basura, menos obvio a la gente que no lo ha usado, es que haciendo que la recolección de basura maneje la memoria simplifica las interfaces entre los componentes (subrutinas,

[10] GC: Recolector de Basura. Referencias: http://www.iecc.com/gclist/GC-faq.html

librerías, módulos, clases) que no necesitan exponer detalles de la administración de memoria.

En el proyecto Mono, actualmente se esta usando el recolector de basura *Boehm*[11], el cual es un recolector de basura para el lenguaje C y C++. Este recolector reemplaza al "malloc" de C, o al "new" de C++. Antes que usar malloc o free para obtener y reclamar memoria, es posible vincular un recolector de basura y permitir reclamar la memoria no utilizada automáticamente.

2.4.3 Multi-threading

Multi-threading es la capacidad de ejecutar simultáneamente diferentes hilos. La programación más tradicional básica es secuencial: después de una instrucción sabemos que se ejecutará la siguiente. Pero si trabajamos con hilos (*threads*) podemos hacer que se ejecuten dos o más flujos de instrucciones simultáneamente, permitiendo por ejemplo, seguir realizando cálculos mientras se espera la entrada por teclado del usuario de algún dato.

2.4.4 PInvoke

Pinvoke (Plataform Invoke)[12] es el mecanismo que se lo usa para envolver las llamadas de los API's de Unix tan bien como si se estuviera hablando a las librerías del sistema.

Con Pinvoke podremos hacer llamadas a librerías, clases o programas en general antiguos, creados en lenguajes diferentes como **C** o **C++** que están fuera de la plataforma .NET. De esta forma se pueden construir librerías

[11] Boehm: Referencias: http://www.hpl.hp.com/personal/Hans_Boehm/gc/
[12] Pinvoke: referencias: http://msdn.microsoft.com/msdnmag/nettop.asp?page=/msdnmag/issues/01/05/com/com0105.asp&ad=ads.ddj.com/msdnmag/ros.htm

intermedias para lenguajes .NET que permita el acceso a funcionalidades implementadas en librerías antiguas.

En este caso se encuentra **GTK#**, el cual hace uso de PInvoke para acceder a las librerías gráficas **GTK** originales y así proporcionar a lenguajes como C# la posibilidad de trabajar gráficamente usando **GTK#**.

2.5 COMPILADOR C# DE MONO (MCS)

MCS es el compilador C# de Mono. Algo muy interesante es que está escrito en el mismo C# y es considerado altamente portable. El compilador C# de Mono es considerado completo hasta este punto y relativamente maduro. MSC esta habilitado para compilarse por si mismo y muchos programas C#. Se lo utiliza para compilar Mono, aproximadamente un millón de líneas del código de C#. Hay un par de áreas que no están cubiertas por el compilador Mono, pero son muy pocas en este punto, se trata de atributos de seguridad.

MCS está escrito en C# y usa todo el potencial de los API's de .Net. MCS corre en Linux (con el runtime de Mono) y en Windows (con el runtime de framework .Net)

MCS no compila código nativo, pero si un tipo de *bytecode*[13], que puede ser procesado por el tiempo de ejecución de Mono, como la siguiente figura nos muestra:

[13] Bytecode: El CIL esta mejor diseñado para ser Compilado en tiempo de ejecución que los Bytecodes de JVM, pero pueden ser interpretados tan fácilmente como si interpretase Bytecode JVM.

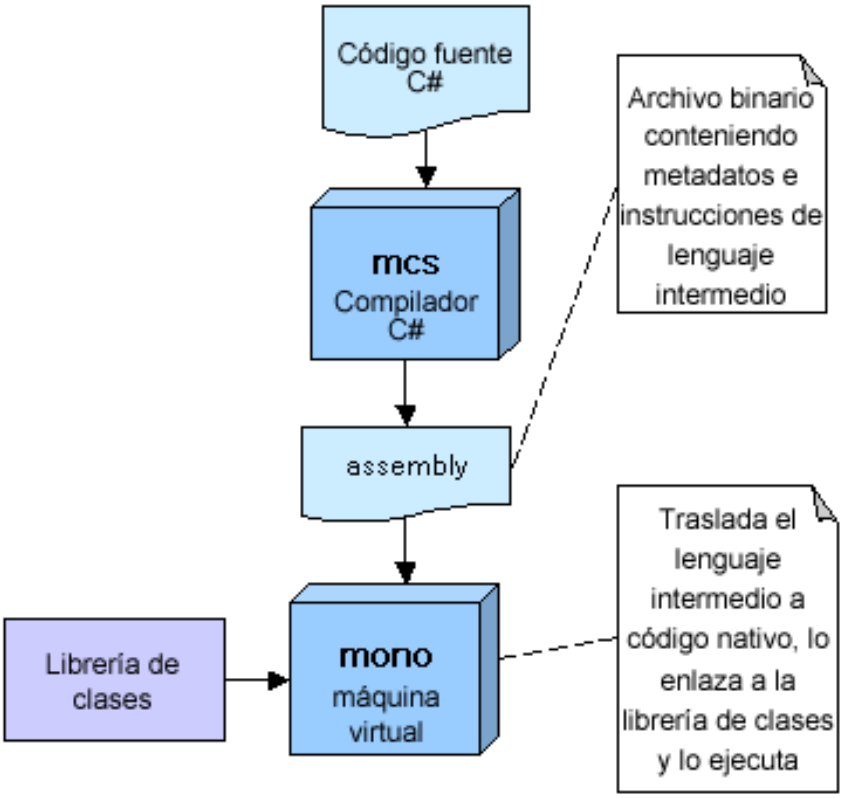


Figura 2.3: Compilación C# de Mono (MCS)

Cuando se reporta un problema, se trata de proveer un test pequeño que puede mostrar el error, este test se lo incluye como una parte de la prueba de regresión de Mono C#. Si este fallo es un error o un aviso de advertencia que no tiene una bandera en el test, entonces se crea un programa simple llamado csXXXX.cs, donde XXXX es el número de código que es usado por el compilador C# que ilustra el problema.

El compilador tiene un número de fases:

- Analizador Léxico: analizador léxico codificado a mano, que provee tokens al parser.
- El Parser (intérprete): el parser es implementado usando *Jay*[14] (se utiliza para portear a Java, y ahora a C#). El parser hace un trabajo

[14] Jay: referencias: http://www.informatik.uni-osnabrueck.de/alumni/bernd/jay/

mínimo y un chequeo sintáctico, y únicamente construye un árbol interpretado.

Cada elemento del lenguaje obtiene sus propias clases. La convención de código usa un nombre en mayúscula para el elemento del lenguaje. Así una clase C# y su información asociada es mantenida en una clase "*Class*", una estructura en una clase "*Struct*" y así sucesivamente. Las declaraciones se derivan de la clase "*Statement*", y las expresiones de la clase "Expr".

- Resolución de la clase padre: antes de la generación de código actual, se necesita resolver las interfaces las clases padre para la interfaz, las clases y las definiciones de estructura.
- Análisis semántico
- Generación de código: La generación de código se la realiza a través del API *System.Reflection.Emit*

2.6 LIBRERÍAS DE CLASE

Mono incluye clases, interfaces, y tipos de valores que aceleran y optimizan el proceso de desarrollo y proveen acceso a la funcionalidad del sistema. Para facilitar la interoperabilidad entre lenguajes los tipos de Mono son flexibles, por tanto son usados por cualquier lenguaje cuyo compilador forme parte de la Especificación de Lenguaje Común (CLS)

La librería de clases provee un comprensivo conjunto de facilidades para el desarrollo de aplicaciones. Primariamente escrita en C#, puede ser usada por cualquier lenguaje, gracias a la Especificación de Lenguaje Común (CLS).

La librería de clases está estructurada en Namespaces, y desplegada en librerías compartidas conocidas como Assemblies.

2.6.1 Namespaces

Namespaces es un mecanismo para agrupamiento lógico de clases similares en una estructura jerárquica. Esto previene conflictos de nombres. La estructura es implementada usando palabras separadas con punto. El namespace de nivel más alto es *System*.

Hay otros namespaces de nivel alto, como son: *Accessibility*, *Windows*. También se pueden crear nuestros propios namespaces, estos pueden tener como prefijo el nombre de nuestra organización, ejemplo: *Microsoft.VisualBasic*.

2.6.2 Assemblies

Assemblies es un empacamiento físico de las librerías de clases. Estos son archivos de extensión .dll, pero no hay que confundirlos con las librerías compartidas de Win32.

Las clases en Mono están organizadas por el assembly al que pertenece.

Estos son los diferentes assemblies:

- *corlib*: Es la librería núcleo
- *System*. Contiene clases fundamentales y clases básicas que definen comúnmente los valores usados y tipos de datos de referencia, eventos y eventos de cabecera, interfaces, atributos, y excepciones de procesamiento. Otras clases proveen servicios soportando la conversión de tipos de datos, métodos de manipulación de parámetros, matemáticas, invocación remota y local, etc.
- *System.Xml* : Provee soporte basado en estándares para procesamiento XML
- *System.Data*: Funcionalidad de acceso a bases de datos (Mono ADO.NET)

- *System.Drawing*: Provee acceso a la funcionalidad básica de gráficos GDI+.
- *System.Web*: Son las clases de ASP.NET
- *System.Web.Services*: Consiste en las clases que habilitan a crear servicios Web XML usando ASP.NET y clientes de servicios Web XML. Los servicios Web XML son aplicaciones que proveen la habilidad de intercambiar mensajes ambiente libremente acoplado usando protocolos estándares como http (*Hypertext Transfer Protocol*), XML (*Extensible Markup Language*), SOAP (*Simple Object Access Protocol*), WSDL (*Web Service Definition Language*). Los servicios Web XML habilitan a construir aplicaciones modulares haciéndolos interoperables con una variedad de implementaciones, plataformas y dispositivos.
- *Microsoft.VisualBasic*: Soporte para aplicaciones de Visual Basic.
- *Windows.Forms*
- *EnterpriseServices*.
- *System.Runtime.Serialization.Formatters.Soap*.
- *System.Security*: Clases de Seguridad XML (Criptografía)

La Librería de clases reside en el módulo MCS en el directorio *Class*. Cada directorio en este directorio representa el ensamblado de donde el código pertenece, y dentro de cada directorio se ha dividido el código basada en el namespace que se estos implementan.

Bibliografía

Referencias WWW

- http://www.go-mono.com
- http://www.go-mono.com:8080/
- http://developer.ximian.com/projects/mono/
- http://www.monohispano.org
- http://www.linuxparatodos.net/linux/art-mono-net.php
- http://www.eurotopfm.com/mono.0.html
- http://congreso.hispalinux.es/congreso2001/actividades/ponencias/garcia/html/modelo.html
- http://www.gotmono.com


CAPÍTULO III
C#
Introducción a la Programación Orientada a Objetos
C#
Tipos
Clases
Constantes, Variables y Propiedades
Operadores
Métodos
Indexadotes
Eventos
Constructores y Destructores
Estructuras
Interfaces
Delegados
Enums
Namespaces y Asemblies

3.1 INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

Los lenguajes computacionales han experimentado una impresionante evolución desde que se construyeron las primeras computadoras. Éstos han permitido escribir programas como una serie de procedimientos que actúan sobre los datos, donde los datos están separados de los procedimientos. La programación Orientada a Objetos trata a los datos y a los procedimientos que operan sobre ellos como un solo objeto.

La programación orientada a objetos surge como un nuevo paradigma que permite acoplar el diseño de programas a situaciones del mundo real, las entidades centrales son los objetos, que son tipos de datos que encapsulan con el mismo nombre estructuras de datos y las operaciones o algoritmos que manipulan esos datos. Algunas de las ventajas de esta aproximación son la reutilización de código, el desarrollo de prototipos en forma sencilla, desarrollo de interfaces gráficas en forma rápida, bases de datos más eficientes.

3.1.1 Fundamentos de la Programación Orientada a Objetos

- La Abstracción permite representar las características esenciales de un objeto, sin preocuparse de las no esenciales.
- La Encapsulación es la propiedad que permite asegurar que el contenido de la información de un objeto está oculto al mundo exterior. La encapsulación permite la división de un programa en módulos. Estos módulos se implementan mediante clases, las clases representan la encapsulación de abstracciones.
- Modularidad: es la propiedad que permite subdividir una aplicación en partes más pequeñas cada una de las cuales debe ser tan independiente como sea posible.

- Jerarquía: es una propiedad que permite ordenar las abstracciones. Las dos jerarquías más importantes son las clases y los objetos. Las clases se relacionan unas con otras por medio de la relación herencia mediante la cual pueden definirse nuevos objetos a partir de los existentes.

3.1.2 Conceptos de la Programación Orientada a Objetos

3.1.2.1 Objeto

Es la unidad que permite combinar datos y funciones que operan sobre esos datos. Dentro de un objeto residen los datos (atributos) de los lenguajes tradicionales, tales como números, arreglos, cadenas y registros que caracterizan el estado del objeto. También residen las funciones o subrutinas (métodos) que operan sobre ellos. Los métodos dentro del objeto son el único medio de acceder a los datos privados de un objeto. No se puede acceder a los datos directamente, los datos están ocultos, y eso asegura que no se pueden modificar accidentalmente por funciones externas al objeto. Los datos y funciones asociados se dicen que están encapsulados en una entidad única o módulo.

Ejemplo de objetos:

a) Objetos físicos: aviones en un sistema de control de tráfico aéreo, casas, parques.

b) Elementos de interfaces gráficas de usuario: ventanas, menús, teclado, cuadros de diálogo.

c) Tipos de datos definidos por el usuario: Datos complejos, Puntos de un sistema de coordenadas

A continuación tenemos la representación gráfica de los objetos:

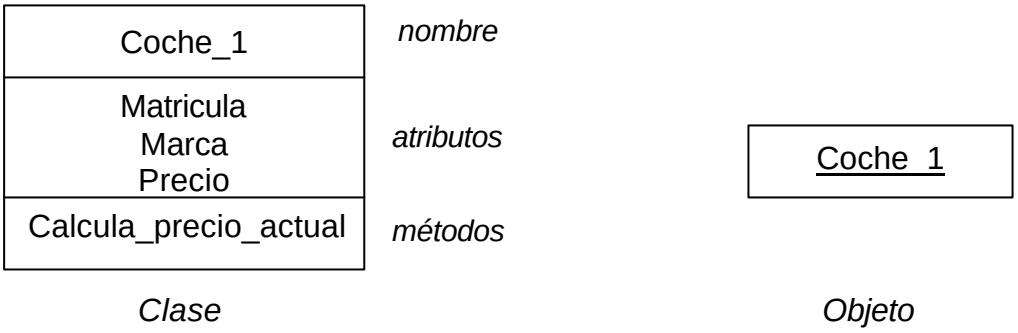


Figura 3.1: Representación gráfica de clases y objetos

3.1.2.2 Métodos y mensajes

Un programa orientado a objetos consiste en un número de objetos que se comunican unos con otros llamando a procedimientos o funciones que reciben el nombre de métodos. Un método es el procedimiento o función que se invoca para actuar sobre un objeto. Los métodos determinan como actúan los objetos cuando reciben un mensaje. Un mensaje es el resultado de cierta acción efectuada por un objeto. El conjunto de mensajes a los cuales puede responder un objeto se le conoce como protocolo del objeto. Por ejemplo, el protocolo de un icono puede constar de mensajes invocados por el clic del botón de un ratón cuando el usuario localiza sobre el icono el apuntador de dicho ratón. De esta forma los mensajes son el único conducto que conectan al objeto con el mundo exterior.

Cuando se ejecuta un programa orientado a objetos ocurren tres sucesos.

1. Los objetos se crean a medida que se necesitan.
2. Los mensajes se mueven de un objeto a otro (o del usuario a un objeto) a medida que el programa procesa información o responde a la entrada del usuario.

3. Cuando los objetos ya no se necesitan, se borran y se libera la memoria.

3.1.2.3 Clase

Es la descripción abstracta de un conjunto de objetos; consta de métodos y datos que resumen características comunes del conjunto de objetos. Se pueden definir muchos objetos de una misma clase. Es decir una clase es la declaración de un tipo de objetos. Cada vez que se construye un objeto a partir de una clase se crea lo que se conoce como instancia de esa clase. Por consiguiente un objeto es una instancia de una clase. La clase tiene dos propósitos: definir abstracciones y favorecer la modularidad.

Existen ciertas clases que se conocen como clases abstractas, estas clases no tiene instancias, pueden no existir en la realidad pero son conceptos útiles, que ocupan un lugar en la jerarquía de clases, actuando como un depósito de métodos y atributos compartidos para las subclases de nivel inferior.

3.1.2.4 Herencia

Es la propiedad que permite a los objetos ser construidos a partir de otros objetos. Dicho de otra forma la capacidad de un objeto para utilizar estructuras de datos y métodos presentes en sus antepasados. El objetivo principal de la herencia es la reutilización, poder utilizar código desarrollado con anterioridad.

La herencia se basa en la división de clases básicas en subclases. Dicha división se fundamenta en el concepto de jerarquía. La herencia supone una clase base y una jerarquía de clases que contiene las clases derivadas. Las clases derivadas pueden heredar el código y los datos de su clase base,

añadiendo su propio código especial y datos, incluso cambiar aquellos elementos de la clase base que necesitan ser diferentes.

No debe confundirse la relación de los objetos con las clases, con la relación de una clase base con sus clases derivadas. Los objetos existentes en la memoria de la computadora expresan las características exactas de su clase. Las clases derivadas heredan características de su clase base, pero añaden otras características propias, nuevas.

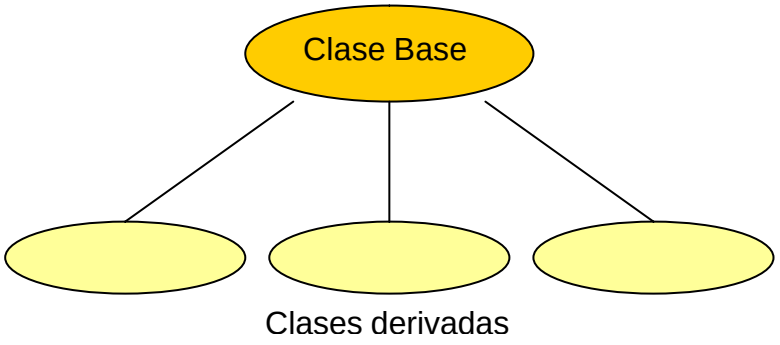


Figura 3.2: Herencia

a) Herencia simple: Una clase puede tener sólo un ascendente. Es decir una subbase puede heredar datos y métodos de una única clase base

b) Herencia múltiple: Una clase puede tener más de un ascendente inmediato, adquirir datos y métodos de más de una clase.

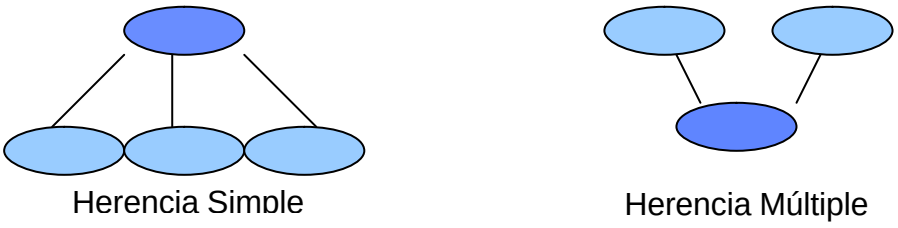


Figura 3.3: Tipos de herencia

La mayor parte de los lenguajes orientados a objetos permiten la herencia simple no así la herencia múltiple porque pueden surgir

ambigüedades. Al aplicar la herencia múltiple si las clases utilizadas para definir una clase nueva tienen un método con el mismo nombre aparecerán problemas de ambigüedad que deberán resolverse con una operación de prioridad que el lenguaje de programación deberá soportar y entender.

3.1.2.5 Polimorfismo

Es el uso de un nombre o un símbolo para representar o significar más de una acción. Los operadores aritméticos de los lenguajes de programación tradicionales son ejemplo de esta propiedad ya que el símbolo + cuando se utiliza con operandos enteros representa un conjunto de operaciones de máquina diferente al conjunto empleado si los operandos son reales.

Otra ilustración sería tener una clase figura que puede aceptar los mensajes dibujar, borrar y mover. Cualquier tipo derivado de una figura es un tipo de figura y puede recibir el mismo mensaje. Cuando se envía un mensaje dibujar, esta tarea será distinta según que la clase sea un triángulo un cuadrado o una elipse.

3.2 C#

C# es un lenguaje diseñado por Anders Heljsberg de Microsoft. C# fue diseñado para ser un lenguaje fácil de usar, donde muchos elementos del lenguaje son similares a Java y C++. Este nuevo lenguaje ha añadido muchos complementos de los lenguajes antiguos.

Microsoft suscribió a C# y la Infraestructura de Lenguaje Común (*Common Language Infrastructure*, CLI) a la ECMA (*European Computer Manufacturers Association*) en el 2000. El ECMA ratifico a C# como un estándar en 2001 como el estándar ECMA-334. LA ISO (*International Standards Organization*) ratificó el estándar ECMA en el 2002.

3.2.1 Estructura Básica

Para empezar con C#, que mejor que con un ejemplo básico para entender la estructura básica de un programa en C# y empezar a tener claros algunos conceptos de la POO (Programación Orientada a Objetos) como por ejemplo que es un espacio de nombres y que es una clase.

El ejemplo sería el siguiente:

```
//Declaración del espacio de nombres
using System;
//Clase de nuestro programa principal
class PrimerEjemplo
    {
      public static void Main()
         {
         // Escribir a la consola
         Console.WriteLine ("Bienvenido a C#");
         }
    }
```

Todo programa que escribamos en C# va a tener una estructura similar a esta en la que declararemos uno o varios espacios de nombres a utilizar (*System*), una clase y el método "*Main*" de esa clase con las sentencias de nuestro programa.

Una vez visto el primer ejemplo, para compilarlo en Linux utilizando la distribución Mono, habría que utilizar el *mcs* (*mono compiler suite*) que es el compilador de C#, implementado siguiendo las especificaciones del lenguaje según el ECMA-334, un proyecto totalmente libre de la arquitectura .NET, que será el encargado de generar los archivos *.exe* en *CIL (Common Infrastructure Language) bytecode* que posteriormente tendrá que ser ejecutado por el *JIT (Just in Time)* de mono. Para ello sería de la siguiente forma:

```
# mcs ejemplo.cs
# mono ejemplo.exe
```

Dando el siguiente resultado:

```
Bienvenido a C#
```

En el ejemplo anterior, ***System*** es un espacio de nombres y con él le estaremos diciendo que podamos usar todos las clases asociadas a ese espacio de nombres, en este caso la clase ***Console*** con el método ***WriteLine*** que es el que se encarga de escribir por pantalla el mensaje que se quiere mostrar.

La clase *PrimerEjemplo* es la que va a contener la definición de datos y métodos que va a usar nuestro programa al ejecutarse. Además de clases veremos que se pueden definir otros tipos diferentes de elementos tales como estructuras e interfaces con los métodos asociados a estos tipos.

Al utilizar el método ***Main*** le estaremos diciendo que nuestro programa empieza ahí y tiene los modificadores ***static*** (sólo se va a usar en esa clase) y ***void*** diciéndole que nuestro método no va a devolver nada, lógicamente podrá tener otros tipos dependiendo de los datos que devuelva y se le podrá pasar parámetros al puro estilo C, pero eso ya lo veremos más adelante.

Ya únicamente queda mostrar por pantalla el resultado, esto se hace utilizando la clase ***Console*** y el método asociado ***WriteLine*** que es el que muestra por pantalla el mensaje de bienvenido. Para hacer referencia a los métodos en ***WriteLine*** en C# se va a hacer con el operador ".", al igual que en Java y a diferencia de C++ donde se utiliza "::".

Para los comentarios se utiliza tanto "*// *" como "*/* esto es un comentario */*" igual que en C++, y todas las sentencias tienen que acabar con ";" y los delimitadores de bloque son "{" y "}".

3.3 TIPOS

C# soporta dos tipos: tipos valor y tipos referencia. Los tipos valor incluyen tipos simples (***char***, ***int*** y ***float***), tipos ***enum*** y tipos ***struct***. Los tipos referencia incluyen tipos clases, tipos interfaces, tipos delegados y tipos ***array***.

Los tipos valor difieren de los tipos de referencia en que las variables de los tipos valor directamente contienen sus datos, mientras que las variables de los tipos referencia almacenan las referencias en objetos.

Con los tipos referencia, es posible para dos variables referenciar al mismo objeto; y hace posible que para operaciones sobre una variable, esto afecte al objeto referenciado por otra variable. Con los tipos valor, cada variable tiene su propia copia de los datos, y no es posible que cuando se realiza operaciones sobre una variable esto afecte a otras.

El siguiente ejemplo muestra la diferencia:

```
using System;
class Class1
{
  public int Value = 0;
}
class Test
{
  static void Main() {
  int val1 = 0;
  int val2 = val1;
    val2 = 123;
    Class1 ref1 = new Class1();
    Class1 ref2 = ref1;
  ref2.Value = 123;
    Console.WriteLine("Values: {0}, {1}", val1, val2);
  Console.WriteLine("Refs: {0}, {1}", ref1.Value, ref2.Value);
  }
}
```

En este ejemplo, el resultado que imprime a pantalla es:

```
Values: 0, 123
Refs: 123, 123
```

3.3.1 Tipos predefinidos

C# provee un conjunto de tipos predefinidos, la mayoría de los cuales serán familiares para los desarrolladores de C y C++.

La siguiente tabla lista los tipos predefinidos e indica como escribir valores literales para cada uno de ellos.

Tipo	Descripción	Rango	Ejemplo
object	El tipo base y esencial de todos los otros tipos.	Cualquier objeto	object o = null;
string	Tipo string; es una secuencia de unidades de código Unicode.	El permitido por la memoria	string s = "hello";
sbyte	8-bit signed integral type	-128 a 127	sbyte val = 12;
short	16-bit signed integral type	[-32.768, 32.767]	short val = 12;
Int	32-bit signed integral type	[-2.147.483.648, 2.147.483.647]	int val = 12;
long	64-bit signed integral type	[-9.223.372.036.854.775.808, 9.223.372.036.854.775.807]	long val1 = 12; long val2 = 34L;
byte	8-bit unsigned integral type	0 a 255	byte val1 = 12;
ushort	16-bit unsigned integral type	[0, 65.535]	ushort val1 = 12;
uint	32-bit unsigned integral type	[0, 4.294.967.295]	uint val1 = 12; uint val2 = 34U;
ulong	64-bit unsigned integral type	[0-18.446.744.073.709.551.615]	ulong val1 = 12; ulong val2 = 34U;

			ulong val3 = 56L; ulong val4 = 78UL;
float	Single-precision floating point type	[1,5×10-45 - 3,4×1038]	float val = 1.23F;
doubl e	Double-precision floating point type	[5,0×10-324 - 1,7×10308]	double val1 = 1.23; double val2 = 4.56D;
bool	Boolean type; a bool value is either true or false	true, false	bool val1 = true; bool val2 = false;
char	Character type; a char value is a Unicode code unit	[‘\u0000’, ‘\uFFFF’]	char val = 'h';
Deci mal	Precise decimal type with 28 significant digits	[1,0×10-28 - 7,9×1028]	decimal val = 1.23M;

3.3.2 Tipos Array

Los **Arrays** pueden ser uni-dimensionales o multi-dimensionales. Los arrays uni-dimensionales son el tipo más común.

El siguiente ejemplo crea un ***array*** uni-dimensional de valores tipo ***int***, inicializa los elementos del ***array*** y luego los imprime a pantalla.

```
using System;
class Test
{
  static void Main() {
    int[] arr = new int[5];
    for (int i = 0; i < arr.Length; i++)
      arr[i] = i * i;
    for (int i = 0; i < arr.Length; i++)
   Console.WriteLine("arr[{0}] = {1}", i, arr[i]);
  }
}
```

3.4 CLASES

Uno de los puntos más fuertes de C# es su soporte de clases del lenguaje. Va un paso más allá que lenguajes como Java: en C# todos los tipos, incluidos los básicos del sistema, son clases. Por ejemplo, el tipo ***int*** en realidad es un alias de ***Int32*** lo que hace posible cosas como:

```
using System;

class TestTipos {
        public static void Main (string[] args) {
                Console.WriteLine (3.ToString());
        }
}
```

, lo cual imprime por consola el número 3 transformado a un dato de tipo ***string***.

Las declaraciones de clases definen nuevos tipos de referencia. Una clase puede heredarse de otra y puede implementar interfaces.

Los miembros de una clase pueden incluir: constantes, campos, métodos, propiedades, eventos, indexadores, operadores, constructores instanciados, constructores estáticos, destructores y declaraciones de tipos anidados.

3.4.1 Reglas de visibilidad

En la mayoría de los lenguajes con soporte de clases, tenemos dos tipos de visibilidad principales: pública y privada; pero además tenemos otras reglas como son: protegida, interna e interna protegida.

- La *pública* indica que lo que se permite como público pueda ser usado en cualquier otro lugar.
- La *privada*, que provoca que se tenga un uso restringido a la clase donde se declara. Lo natural es buscar la mayor independencia entre las clases,

para evitar el acoplamiento y facilitar la evolución del código. Por ello, lo más recomendable es hacer privado todo aquello que se pueda.

? Hay situaciones en la que por ejemplo, interesa poder acceder a miembros de una clase desde las clases que la utilizan como base, es decir, desde las clases que heredan de ella. A este tipo de visibilidad se la conoce como *protegida*.

? La *interna* permite acceder a otras zonas del programa actual que son permitidas y,

? Una última regla mixta, *protegida – interna,* permite la visibilidad dentro del mismo programa o de clases de otros programas pero que heredan de la clase base con esta visibilidad.

Las clases que se declaran dentro de otra tienen acceso a todos los elementos de la clase que la contienen. Veámoslo con un ejemplo:

```
using System;
class TestTipos {
  private int contador;

   public static void Main (string[] args) {
           Console.WriteLine (3.ToString());
    }
   class Interna {
         public void Prueba () {
            TestTipos  t = new TestTipos();
             // Acceso a miembro privado permitido
             int i = t.contador;
           }
    }
   }

class TestTipos2 {
   private void Prueba () {
      TestTipos  t = new TestTipos();
       int i = t.contador;
      //Error de compilación porque es privada en la clase TestTipos
    }
}
```

3.5 CONSTANTES, VARIABLES Y PROPIEDADES

Dentro de una clase a parte de los métodos que definen los que hace la clase, tenemos los elementos que definen el estado de una clase y sus instancias, que son las constantes y las variables. Un tipo especial de variables son las propiedades que facilitan mucho el seguimiento del contenido de las variables. Vamos con un ejemplo de variables y constantes y luego mostraremos las propiedades.

```
using System;

class TestVariables {
        private static int contador;
    const int version = 1;

        public static void Main (string[] args) {
          string nombre = "El aprendiz de C#";
                Console.WriteLine (nombre);
          contador = 20;
          Console.WriteLine (contador.ToString());
          Console.WriteLine (version.ToString());
        }
}
```

En este ejemplo vemos una variable de la clase, *contador*, que es de tipo ***static***. Dentro del método principal de la clase ***Main*** declaramos otra variable, en este caso de tipo ***string***. Vemos que aquí no es necesario especificar que la variable *nombre* es de tipo ***static*** ya que al ser declarada dentro de un método ***static*** lo es de forma automática.

Vemos como el tratamiento unificado de todo como clases hacen que incluso las constantes sean objetos, lo que permite imprimirlas también de forma sencilla, por ejemplo.

Al pasar variables a funciones, las podemos pasar por valor o por referencia. Por defecto, las variables de tipos básicos y las estructuras se

pasan como copias, es decir, por valor. Las instancias de clases, los objetos, se pasan por referencia.

Vamos ahora con el uso de propiedades, una de las características más potentes de las variables de C#. Al tener controlado el acceso a la propiedad, podemos por ejemplo emitir eventos cuando cambie y así informar a otras clases que pudieran estar interesadas.

```
using System;

class TestProperties {
      private static string clave;
      public string Clave {
            get
            {
                  Console.WriteLine ("Acceso a la propiedad clave");
                  return clave;
            }
            set
            {
                  Console.WriteLine ("Cambio del valor de clave");
                  clave = value;
            }
      }
}
```

```
class Test {
            public static void Main (string[] args) {
              TestProperties tp = new TestProperties();
              string c = "ClaveClave";
              tp.Clave = c;
              Console.WriteLine (tp.Clave);
            }
}
```

En realidad, lo que se hace es declarar una variable privada de forma que no se puede acceder de forma directa, y se crean dos métodos públicos que permiten acceder al contenido de la variable y modificarla. Si no queremos que se pueda modificar la variable, no incluimos el método *set* y ya tendríamos propiedades de sólo lectura.

3.6 OPERADORES

Los operadores son símbolos que permiten realizar operaciones con uno o más datos, para dar un resultado. El ejemplo clásico y obvio de operador es el símbolo de la suma (**+**), aunque hay otros muchos.

Vamos a hacer un repaso a los tipos de operadores que tenemos en C#:

? Operadores Aritméticos: Son la suma (**+**), resta (**-**), producto (*****), división (***/***) y módulo (**%**).

Es interesante saber que a las operaciones aritméticas les podemos añadir otros dos operadores, ***checked*** y ***unchecked***, para controlar los desbordamientos en las operaciones. La sintaxis es esta:

```
variable1 = checked (34+4);
variable1 = unchecked (34+4);
```

Si en una operación regida por ***checked*** se produce un desbordamiento, dará un error en tiempo de compilación si los operadores son constantes, o lanzará una excepción ***System.OverflowException*** si son variables. En cambio, si la operación es ***unchecked***, cuando se produce un desbordamiento nos devolverá el resultado truncado, para que quepa en el resultado esperado.

? Operadores Lógicos: Son ***and*** (**&&** y **&**), ***or*** (**||** y **|**), ***not*** (**!**) y ***xor*** (^).

La diferencia entre **&&** y **&**, y entre **||** y **|** es que **&&** y **||** hacen lo que se llama "evaluación perezosa": si evaluando sólo la primera parte de la operación se puede deducir el resultado, la parte derecha no se evaluará. Es decir, si tenemos por ejemplo:

```
false && (expresión)
```

El resultado de esta operación siempre será ***false***, y *(expresión)* ni siquiera se evalúa. De igual forma, si tenemos:

```
true || (expresión)
```

, el resultado será **true**, y la parte derecha nunca se evaluará.

- Operadores relacionales: igualdad (**==**), desigualdad (**!=**), "mayor que" (**>**), "menor que" (**<**), "mayor o igual que" (**>=**) y "menor o igual que" (**<=**)
- Operadores de Manipulación de Bits: Tenemos las siguientes operaciones: and (**&**), or (**|**), not (**~**), xor (**^**), desplazamiento a la izquierda (**<<**), y desplazamiento a la derecha (>>). El desplazamiento a la izquierda rellena con ceros. El desplazamiento a la derecha, si se trata de un dato con signo, mantiene el signo. Si no, rellena con ceros.
- Operadores de Asignación: El operador básico de asignación es **=**. Además, tenemos las clásicas abreviaturas **+=**, **-=**, ***=**, **/=**, **&=**, **|=**, **^=**, **<<=** y **>>=**

  También tenemos operadores de incremento (**++**) y decremento (**--**), que incrementan en una unidad el valor de la variable sobre la que se aplican.
- Operador Condicional: Es el único operador de C# que tiene tres operandos. Su sintaxis es esta:

```
<condición> ? <expresión1> : <expresión2>
```

  Quiere decir que si la condición es ***true***, se evalúa *expresión1*, y si es falsa, se evalúa *expresión2*. Se ve más claro con un ejemplo:

```
b = (a>0) ? a : 0;
```

  Esto quiere decir que si *a* es mayor que *0*, la expresión será *b = a*, y si *a* no es mayor que *0*, la expresión será *b = 0*.

Ojo: No confundir con un ***if***. Este operador devuelve un valor, mientras que el ***if*** es solamente una instrucción.

- Operadores de Delegados: Para añadir métodos a un delegado se hace con **+** y **+=**, y para quitárselos, con **-** y **-=**.

- Operadores de Acceso a Objetos: El operador para acceder a los miembros de un objeto es el punto. Así esta expresión:

```
A.metodo1 ();
```

, nos permite acceder al método "*metodo1*" del objeto *A*.

- Operadores de Punteros: Tenemos varios operadores. Para acceder a la dirección de memoria a la que apunta el puntero, lo hacemos con *&puntero*. Para acceder al contenido de la dirección de memoria, tenemos **puntero*. Si lo que referencia el puntero es un objeto, podemos acceder a sus miembros con *puntero->miembro*.

- Operadores de Obtención de Información sobre Tipos: Para averiguar el tipo de una variable, usamos el operador *sizeof (variable)*. Nos devolverá un objeto de tipo ***System.Type***. Si queremos hacer una comparación, usamos algo como esto:

```
(expresion) is nombreTipo
```

, que, como es lógico, nos devolverá un ***true*** o un ***false***.

- Operadores de Conversión: Para convertir el tipo de un objeto en otro, precedemos el objeto que queremos cambiar con el tipo al que queremos convertir, entre paréntesis, de esta forma:

```
variable1 = (int)
```

De esta forma, *variable2* será tratada como si fuera un dato de tipo ***int***, aunque no lo sea.

3.7 MÉTODOS

Un método es un miembro que implementa una acción que puede ser realizada por un objeto o clase.

Los métodos tienen una lista de parámetros formales (posiblemente vacía), un valor de retorno (a menos que el tipo de retorno del método sea ***void***), y pueden ser estáticos o no estáticos. Los métodos estáticos son accedidos a través de las clases. Los métodos no estáticos los cuales son también llamados métodos de instancia, son accedidos a través de instancias de la clase.

```
using System;
public class Stack
{
  public static Stack Clone(Stack s) {...}
  public static Stack Flip(Stack s) {...}
  public object Pop() {...}
  public void Push(object o) {...}
  public override string ToString() {...}
  ...
}

class Test
{
  static void Main() {
    Stack s = new Stack();
    for (int i = 1; i < 10; i++)  s.Push(i);
    Stack flipped = Stack.Flip(s);
    Stack cloned = Stack.Clone(s);
    Console.WriteLine("Original stack: " + s.ToString());
    Console.WriteLine("Flipped stack: " + flipped.ToString());
    Console.WriteLine("Cloned stack: " + cloned.ToString());
   }
}
```

Este ejemplo muestra una clase (*Stack*) que tiene algunos métodos estáticos (*Clone* y *Flip*) y algunos métodos de instancia (*Pop*, *Push*, y *ToString*).

3.8 INDEXADORES

Hemos visto, en la parte donde se trata a las propiedades, que podemos acceder a una variable privada de una clase a través de eventos que nos permiten controlar la forma en la que accedemos a dicha variable.

Los indexadores nos van a permitir hacer algo parecido, pero dando un paso más allá: acceder a una clase como si se tratara de un ***array***. Lo vemos de forma más sencilla con un ejemplo:

```
using System;
class PruebaIndexadores
{
  private int[] tabla = {1, 2, 3, 4};

  public int this [int indice]
  {
    get
    {
      Console.WriteLine ("La posicion {0} de la tabla tiene el valor {1}",
indice, tabla[indice]);
      return tabla[indice];
    }
    set
    {
      Console.WriteLine ("Escrito el valor {0} en la posición {1} de la tabla",
value, indice);
      tabla[indice] = value;
    }
  }
}
```

Tenemos una clase *PruebaIndexadores* en la que hay un ***array*** llamado "*tabla*", declarado como privado, por lo que no podremos acceder a él desde fuera de nuestra clase. Pero hemos declarado también un indexador (*public int this [int indice]*), que nos permitirá acceder a él de forma más controlada.

Para probar esta clase, creamos otra clase con un punto de entrada (*public static void Main ()*), que será donde hagamos las pruebas.

Primero creamos un objeto de la clase *PruebaIndexadores*:

```
PruebaIndexadores obj = new
PruebaIndexadores ();
```

Luego accedemos a una posición del indexador:

```
int a = obj[3];
```

Esta línea lo que hace es llamar al indexador, pasándole como parámetro el índice, en este caso 3. Al ser una consulta de lectura, se ejecuta el código que haya en la parte "*get*" del indexador. Una vez ejecutado, lo que nos aparece por pantalla es esto: *La posición 3 de la tabla tiene el valor 4*

3.9 EVENTOS

Mientras que antiguamente la interacción entre un programa y el usuario se limitaba a que éste introdujese datos en determinados momentos de la ejecución del primero, en los últimos años este modelo está siendo relegado por un crecimiento exponencial de las aplicaciones GUI (*Graphical User Interface o Interfaz Gráfica de Usuario*). En esta nueva aproximación conocida como Programación Orientada a Eventos el programa queda a la espera de que el usuario vaya haciendo tal o cual cosa sin tener casi ningún control sobre el orden en el que se producirán los sucesos o más aún, ni cuales son los que darán. Más aún este tipo de programación proporciona una asincronía muy útil gracias al uso de las devoluciones de llamada, tal y como se vio con los delegados.

En nuestro caso estamos de suerte porque el lenguaje en sí soporta eventos. Sin entrar en detalles sobre qué son exactamente, veremos como usarlos. Podemos decir que los eventos son una especie de mensajes que lanza el entorno de ejecución de manera que un objeto pueda avisar cuando ha sucedido algo de lo que quiera avisar a otros objetos.

En .Net se usa un mecanismo conocido como de publicación o suscripción. Así que lo que tendremos será una clase que publica un evento al que, aquellas clases que quieran estar al tanto de cuando ha sido lanzado ese evento se subscriben. De hecho, es una práctica común no lanzar el evento si no hay ninguna clase subscrita.

Veamos el procedimiento general a seguir.

1. Crear la clase que contendrá el evento, la que lo publica (la que lo puede lanzar). Supongamos que el evento que queremos lanzar se llama ***OnButtonClicked***. Se puede poner un nombre genérico como *MyEvent*. Consecuentemente hay que cambiar todos los ***OnButtonClicked*** por *MyEvent* si se hace esto eso.
2. Definimos el manejador del evento. Es el código que se asocia con el evento en tiempo de ejecución y que se invoca cuando la clase que se suscribe recibe la notificación del evento. Es un delegado de tipo ***void*** y que recibe dos parámetros: el primero un objeto, el segundo una instancia de la clase que guarda la información del evento. Esta clase tiene que derivar de *EventArgs*.

```
[modificadores de acceso] delegate void
OnButtonClickedEventHandler (object source,
OnButtonClickedEventArgs e);
```

3. Declaramos el evento con el tipo del delegado antes definido

```
[modificadores de acceso] event OnButtonClickedEventHandler
OnButtonCliked;
```

4. Definimos el método que puede lanzar el evento y establecemos los valores para el objeto que guarda la información del evento. Esto es código en la clase que publica el evento y es el código que se encarga

de lanzar el evento con la información adecuada en el momento adecuado.

5. Creamos la clase que guarda la información del evento derivándola de *EventArgs*:

```
class OnButtonClickedEventArgs : EventArgs {
    // código
    // al menos un constructor y conveniente que tenga propiedades
para   acceder a las variables miembro.
    }
```

6. Creamos la clase que se suscribe al evento

7. Suscribimos la clase con:

```
ClaseQuePublicaElEvento.OnButtonClicked += new
OnButtonClickedEventHandler (callback);
```

Donde *callback* es la función de devolución de llamada (retrollamada o callback) que se quiera llamar cuando se recibe la notificación del evento. Esa función debe estar en la clase que se subscribe al evento.

Como el operador que se usa para las subscripciones es del tipo **+=** podemos suscribir varios manejadores de evento a un mismo evento. Y así añadir tantos *callbacks* como queramos ya que no se sobrescriben los unos a los otros (lo que pasaría si se hubiese usado el operador **=**) si no que se añaden. También podemos de - suscribirnos de un evento mediante el operador **-=**. Para ello no hay más que escribir lo mismo pero con un **-** en vez de un **+**:

3.10 CONSTRUCTORES Y DESTRUCTORES

Un constructor es un método especial que se ejecuta cada vez que se crea un objeto de una clase. Se utiliza para inicializar variables, y todo lo que corresponda hacer cuando creamos un objeto. Su sintaxis es muy sencilla, y la vemos con un ejemplo:

```
using System;
class Persona
{
  string nombre;
  int    edad;
  string nif;

  public void Persona (string nombre, int edad, string nif)
  {
    this.nombre = nombre;
    this.edad   = edad;
    this.nif    = nif;
  }
}
```

Cuando nosotros creemos desde otro lugar un objeto de la clase Persona, lo haremos de esta forma:

```
Persona pablo = new Persona ("Pablo Landeta", 23, "111111-m");
```

Lo que hará el compilador de C# es buscar el constructor que se adecue a los parámetros que le llegan, y ejecutarlo como si fuera un método más. Es muy importante cuando decimos que busca el constructor que se adecue a los parámetros, ya que podemos tener varios constructores, con parámetros distintos. Dependiendo de la llamada que se haga en el ***new***, usaremos un constructor u otro.

Nos puede interesar, en el caso de que tengamos varios constructores, que unos se llamen a otros, para no tener que rescribir código que vayamos a ejecutar siempre. Lo podemos hacer de la siguiente forma:

```
using System;
class Persona
{
  string nombre;
  int    edad;
  string nif;

  public Persona (string nombre, int edad): this (nombre, edad, "11111-q")
  {
  }

  public Persona (string nombre, int edad, string nif)
  {
    this.nombre = nombre;
    this.edad   = edad;
    this.nif    = nif;
  }
}
```

El primer constructor permite que cuando creemos un objeto de esta forma:

```
Persona pablo = new Persona ("Pablo Landeta", 23);
```

En realidad lo que haga sea llamar a otro constructor, pasándole a alguno de sus parámetros unos valores por defecto. En este caso, lo que hará será crear un objeto "*pablo*" con nombre "*Pablo Landeta*", de edad 23, y pasándole una variable *NIF* por defecto que valga "*11111-q*".

Un detalle importante: los constructores NO SE HEREDAN. Es decir, si nosotros creamos una clase *Estudiante* que herede de la clase *Persona*, cuando creemos un objeto de *Estudiante*, el único constructor al que se llamará será al de *Estudiante*. En ningún caso se ejecutará código del constructor de *Persona*.

De forma intuitiva, el concepto de destructor en C# puede no tener mucho sentido, ya que tenemos el recolector de basura, que se encargará de hacer la labor del destructor de forma automática. Pero, cuando tengamos clases más complejas, sí puede venirnos bien. Podemos querer asegurarnos, por ejemplo, de que una clase que maneja conexiones de red las deja correctamente cerradas antes de destruirse.

Los destructores se declaran de la siguiente forma:

```
using System;
class Persona
{
  ~Persona()
  {
    Console.WriteLine("Destruido el objeto de la clase Persona");
  }
}

class Estudiante:Persona
{
  ~Estudiante()
  {
    Console.WriteLine("Destruido el objeto de la clase Estudiante");
  }

  public static void Main()
  {
    new Estudiante();
  }
}
```

Lo primero que se ve aquí es que en ningún momento destruimos explícitamente los objetos. De eso se encarga el recolector de basura. Lo único que tenemos que programar es que, cuando el recolector decida que hay que destruir un objeto, lo haga de la forma adecuada.

¿Por qué en este ejemplo he utilizado herencia? Porque, a diferencia de los constructores, los destructores sí que se heredan. Si nosotros creamos un

objeto de la clase *Estudiante*, que hereda de *Persona*, a la hora de destruirlo se ejecutará el destructor de *Estudiante*, y luego el de *Persona*.

3.11 ESTRUCTURAS

Las estructuras son similares a las clases ya que pueden agrupar un conjunto de variables, tener métodos e incluso implementar interfaces. Pero las estructuras a diferencia de las clases, utilizan variables que se usan por valor y no por referencia, y no se puede utilizar la herencia en las estructuras.

Por ejemplo, el uso de una estructura en vez de una clase, puede hacer la diferencia en el número de asignaciones de memoria realizadas en la corrida. El siguiente programa crea e inicializa un ***array*** de 100 puntos. Con el programa implementado como una clase, 101 objetos separados son instanciados (uno para el ***array*** y uno para cada uno de los 100 elementos)

```
class Point
{
  public int x, y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
class Test
{
  static void Main() {
    Point[] points = new Point[100];
    for (int i = 0; i < 100; i++)
   points[i] = new Point(i, i*i);
  }
}
```

Si se implementa como una estructura, únicamente un objeto es instanciado: el del ***array***. Las instancias de este ejemplo son asignadas en línea mediante el ***array***. Usando estructuras en vez de clases puede hacer que

una aplicación corra más rápido o utilice menos memoria, al igual que cuando se pasa una instancia de una estructura por valor causa una copia de la estructura a ser creada.

```
struct Point
{
  public int x, y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

3.12 INTERFASES

Una interfase define un contrato. Una clase o estructura que implemente una interfase debe adherirse a este contrato. Las interfaces pueden contener métodos, propiedades, eventos e indexadotes como miembros.

El siguiente ejemplo nos muestra una interfase que contiene un indexador, un evento *E*, un método *F*, y una propiedad *P*.

```
interface ejemplo
{
  string this[int index] { get; set; }
  event EventHandler E;
  void F(int value);
  string P { get; set; }
}
public delegate void EventHandler(object sender, EventArgs e);
```

3.13 DELEGADOS

Los delegados son un tipo especial de clases que derivan de ***System.Delegate***. Son especiales en el sentido que no se crean utilizando los mecanismos de herencia habituales, sino que cuentan con su propia forma de declaración.

Respecto a la herencia (a diferencia de C++ y al igual que Java), C# sólo admite herencia simple de clases ya que la múltiple provoca más complicaciones que facilidades y en la mayoría de los casos su utilidad puede ser simulada con facilidad mediante herencia múltiple de interfaces.

Los delegados tienen como objetivo almacenar referencias a métodos de otras clases, de tal forma que, al ser invocados, ejecutan todos estos métodos almacenados de forma secuencial. Si ya tenemos conocimientos previos de C/C++, apreciaremos la similitud con los punteros a funciones. Pero los delegados tienen una serie de ventajas añadidas.

Los delegados pueden almacenar métodos estáticos o instanciados, indiferentemente. Además los tipos de estos métodos son comprobados para evitar errores de programación. Es decir, un delegado no almacenará referencias de un método si éste no tiene la misma estructura que el delegado. Esto es así porque cuando se declara un delegado, estamos indicando también como van a ser los métodos que podrá manejar.

Veamos esto con un ejemplo. Aquí podemos ver la declaración de nuestro primer delegado:

```
public void delegate MiDelegado(int entero, string cadena);
```

La sentencia anterior ha declarado al delegado *MiDelegado*. Sólo podrá guardar referencias a métodos que no devuelvan nada (***void***) y que reciban como parámetros un entero y una cadena. Cualquier intento de guardar métodos no compatibles provocará una excepción.

Una vez tenemos declarado nuestro delegado, ya podemos crear instancias del mismo. Veamos este sencillo ejemplo:

```
using System;
class Ejemplo {
  delegate void MiDelegado (int a);
  void Imprime (int entero) {
    Console.WriteLine ("Llamada a Ejemplo.Imprime({0})", entero);
  }

  public static void Main() {
    Ejemplo ejemplo = new Ejemplo ();
    MiDelegado foo;
    foo  = new MiDelegado (ejemplo.Imprime);
    foo (10);
  }
}
```

Tras crear una instancia de nuestra clase *Ejemplo*, creamos otra de nuestro delegado. Posteriormente insertamos en el delegado una referencia al método Imprime, que pertenece a la instancia de nuestra clase. Finalmente convocamos al delegado a que ejecute los métodos que contenga. Si lo compiláramos obtendríamos esto:

```
Llamada a Ejemplo.Imprime(10)
```

Otra cuestión a tener en cuenta cuando programemos con delegados, es que éstos no tienen en cuenta la visibilidad de los métodos. Esto permite llamar a métodos privados desde otros si ambos tienen acceso al delegado. Es decir, imaginemos que una clase guarda en un delegado referencia a uno de sus métodos privados. Si desde otra clase que tenga acceso al delegado (pero

no al método privado) se convoca a éste, se ejecutará ese método. En verdad no se está violando la privacidad del método, porque no es la dase quien lo convoca, sino el delegado, que sí tiene acceso al mismo.

3.14 ENUMS

Una declaración de tipo *enum*, define un nombre de tipo para un grupo relacionado de constantes simbólicas. *Enum* es usado para escenarios de "múltiple opción", donde la decisión de ejecución se realiza desde un número fijo de opciones que son conocidas en tiempo de compilación.

El siguiente ejemplo muestra un *enum Color* y un método que usa este *enum*. El método *Fill* lo limpia para que la figura pueda ser llenada con uno de los colores dados.

```
enum Color
{
  Red,
  Blue,
  Green
}
class Shape
{
  public void Fill(Color color) {
  switch(color) {
   case Color.Red:
        ...
        break;
   case Color.Blue:
        ...
        break;
case Color.Green:
        ...
        break;
      default:
        break;
    }
  }
}
```

El uso de *enums* es superior al uso de constantes enteras (como es común en los lenguajes que no implementan *enum*) porque el uso de *enums* hace el código más legible y auto documentado.

3.15 NAMESPACES y ASSEMBLIES

Es muy común en el mundo real, que las aplicaciones consistan en algunas diferentes piezas, cada una compilada separadamente. Por ejemplo, una aplicación corporativa puede depender de algunos diferentes componentes, incluyendo algunos desarrollados internamente y algunos otros adquiridos de vendedores de software independientes.

Los ***Namespaces*** y los ***Assemblies*** permiten este sistema basado en componentes. Los ***Namespaces*** proveen un sistema de organización lógica.

Los ***Assemblies*** son usados para el empacamiento y desplegamiento físico. Un ***assembly*** puede contener tipos, el código ejecutable usado para implementar estos tipos, y referencias a otros ***assemblies***.

Para demostrar esto, tenemos un ejemplo que se lo ha dividido en dos partes: una librería de clases que provee un mensaje y una aplicación de salida a consola que muestra el mensaje.

La primera parte es la librería de clase que contiene una clase simple llamada *HelloMessage*:

```
namespace CSharp.Introduction
{
 public class HelloMessage
  {
    public string Message {
      get {
    return "hello, world";
      }
    }
  }
}
```

Este ejemplo muestra la clase *HelloMessage* en un ***namespace*** llamado *CSharp.Introduction*. La clase *HelloMessage* provee una propiedad solo de lectura llamada *Message*.

Los **Namespaces** pueden anidarse, y la declaración:

```
namespace CSharp.Introduction
{.}
```

, es una abreviatura de dos niveles de un ***namespace*** anidado, así:

```
namespace CSharp
{
  namespace Introduction
  {.}
}
```

La siguiente parte es sacar a consola el mensaje. La forma acertada es usar la directiva ***namespace***:

```
using CSharp.Introduction;
class HelloApp
{
  static void Main() {
    HelloMessage m = new HelloMessage();
  System.Console.WriteLine(m.Message);
  }
}
```

Muestra el uso de la directiva ***namespace***, la cual se refiere al ***namespace*** *CSharp.Introduction.*

C# también permite la definición y uso de aliases. Usando la directiva *alias*, define un alias para un tipo. Estos aliases pueden ser de gran ayuda en situaciones en donde ocurren colisiones de nombres entre dos librerías de clase, o cuando un pequeño número de ***namespaces*** muy largos han sido usados. El ejemplo:

```
using MessageSource = CSharp.Introduction.HelloMessage;
```

Muestra el uso de la directiva *alias*, que define a *MessageSource* como un alias de la clase *HelloMessage*.

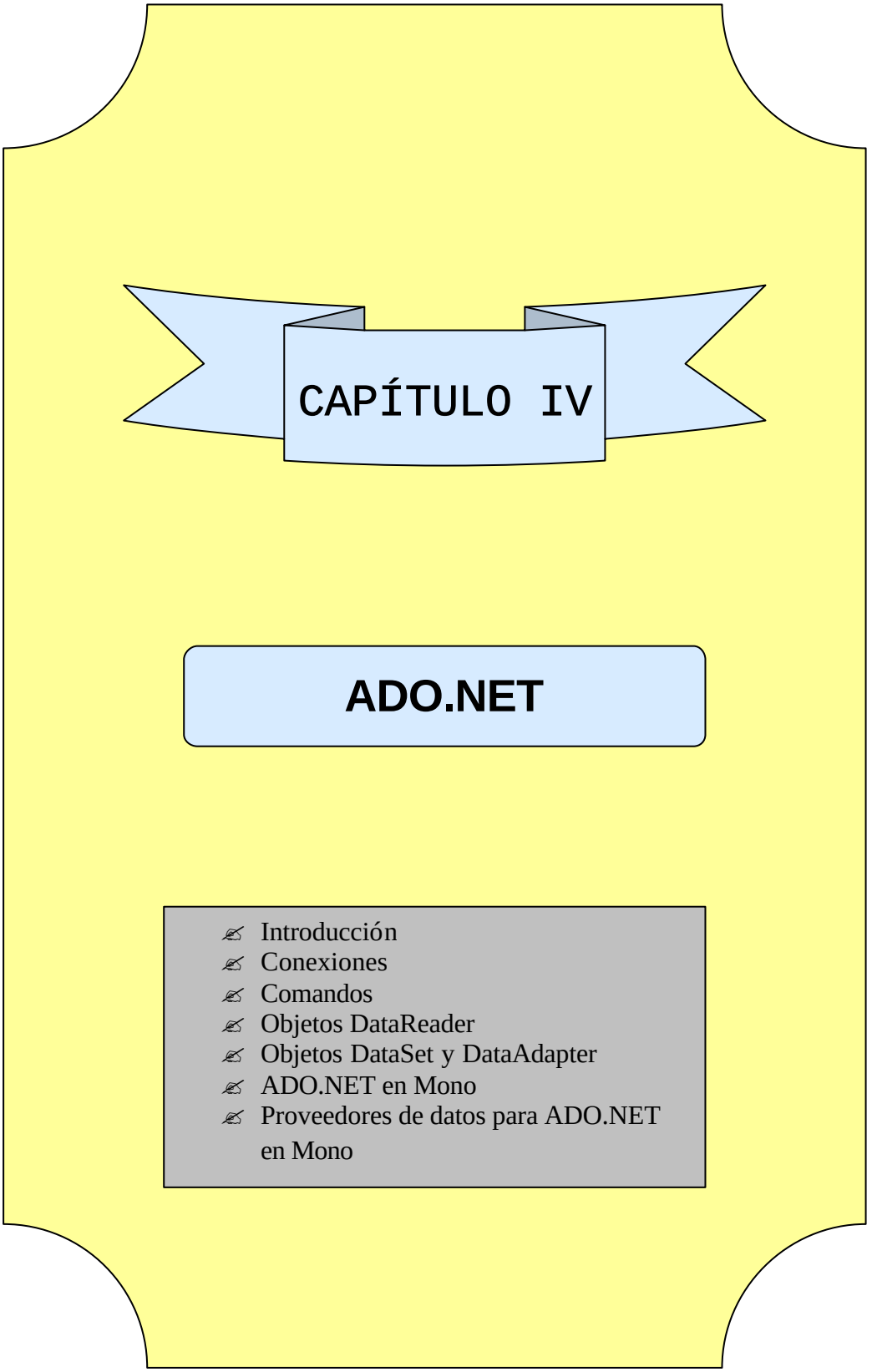
Bibliografía

Libros

- ARCHER,Tom, “A Fondo C#”, Editorial Mc Graw-Hill, 2001
- RODRIGUEZ, Miguel, “Desarrollo de Aplicaciones .NET con Visual C#”, Editorial Mc Graw-Hill, 2002

Referencias WWW

- http://www.clikear.com/manual_csharp/c10.asp
- http://codorniz.arcos.inf.uc3m.es/msaa/documentos/Como2.html
- http://www.informit.com/isapi/product_id~{E677E33F-8FA6-4543-96E4-259920DA7106}/element_id~{55D2B30E-1C1A-4B95-88EA-ACBE727F5613}/st~{EA7C8D03-4995-402D-B085-06E000F897B8}/content/articlex.asp
- http://www.oreilly.com/catalog/csharpnut/chapter/ch01.html


CAPÍTULO IV
ADO.NET
Introducción
Conexiones
Comandos
Objetos DataReader
Objetos DataSet y DataAdapter
ADO.NET en Mono
Proveedores de datos para ADO.NET en Mono

4.1 INTRODUCCION

ADO.NET es una evolución del modelo de acceso a datos de ADO(*Access Data Objects*) que controla directamente los requisitos del usuario para programar aplicaciones escalables. Se diseñó específicamente para el Web, teniendo en cuenta la escalabilidad, la independencia y el estándar XML.

ADO.NET utiliza algunos objetos ADO, como **Connection** y **Command**, y también agrega objetos nuevos: **DataSet**, **DataReader** y **DataAdapter**.

- Objetos **Connection**: Se lo utiliza para conectar con una base de datos y administrar las transacciones en una base de datos.
- Objetos **Command**: Se lo utiliza para emitir comandos SQL a una base de datos.
- Objetos **DataReader**: Proporcionan una forma de leer una secuencia de registros de datos sólo hacia delante desde un origen de datos.
- Objetos **DataSet**: Se lo utiliza para almacenar datos sin formato, datos XML y datos relacionales, así como para configurar el acceso remoto y programar sobre datos de este tipo.
- Objetos **DataAdapter**: Se lo utiliza para insertar datos en un objeto **DataSet** y actualizar datos de la base de datos.

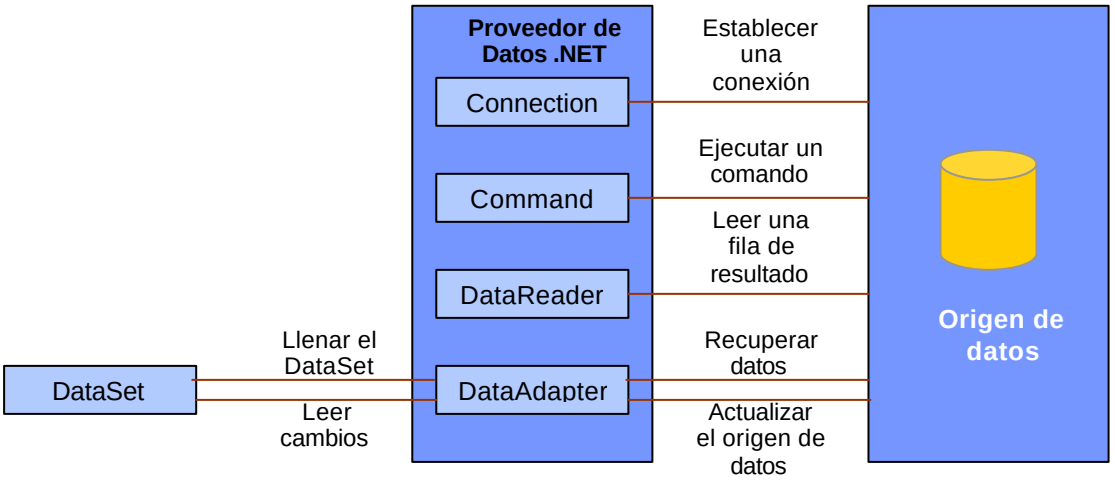


Figura 4.1: Objetos de ADO.NET

La diferencia más importante entre esta fase evolucionada de ADO.NET 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 el objeto **DataSet** como un conjunto de registros que siempre está desconectado y que no sabe nada sobre el origen y el destino de los datos que contiene. Dentro de un objeto **DataSet**, de la misma manera que dentro de una base de datos, hay tablas, columnas, relaciones, restricciones, vistas, etc.

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. 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 de origen 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. Funciona sin tener conocimiento del origen de los datos y representa a los datos que contiene como colecciones y tipos de datos. Independientemente del origen de los datos del objeto **DataSet**, éstos se manipulan mediante el mismo conjunto de API's 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. La función del proveedor administrado es conectar, llenar y almacenar el objeto **DataSet** desde almacenes de datos (o viceversa).

4.2 CONEXIONES

Para establecer la comunicación con bases de datos, se utilizan las conexiones y se representan mediante clases específicas de proveedor. Los comandos viajan por las conexiones y devuelven conjuntos de resultados en forma de secuencias que puede leer un objeto **DataReader** o que se pueden insertar en un objeto **DataSet**.

En el siguiente ejemplo se muestra la forma de crear un objeto de conexión utilizando C#. Las conexiones se pueden abrir explícitamente mediante llamadas al método **Open** de la conexión; también se pueden abrir implícitamente al utilizar un objeto **DataAdapter**.

```
using System;
using System.Data.SqlClient;
public class ado1
{
  public static void Main()
  {
    ado1 myado = new ado1();
    myado.Run();
  }
  public void Run()
  {
    SqlConnection mySqlConnection = new
SqlConnection("server=(local)\\NetSDK;Trusted_Connection=yes;database=bdd");
    try {
      mySqlConnection.Open();
      Console.WriteLine("Conexin con {0} abierta",
mySqlConnection.ConnectionString);
      // Cerrar la conexión explicitamente
      mySqlConnection.Close();
      Console.WriteLine("Conexión cerrada.");
    }
    catch
    {
      Console.WriteLine("No se pudo abrir la conexión con {0}",
mySqlConnection.ConnectionString);
    }
  }
}
```

4.3 COMANDOS

Los comandos contienen la información que se envía a una base de datos y se representan mediante clases específicas de un proveedor. Un comando podría ser una llamada a un procedimiento almacenado, una instrucción de actualización 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.

En el ejemplo siguiente se muestra como insertar datos en una base de datos utilizando C#.

```
using System;
using System.Data.SqlClient;

public class ado2
{
  public static void Main()
  {
    ado2 myado = new ado2();
    myado.Run();
  }

  public void Run()
  {
    string Message = null;
    SqlConnection myConnection = new
SqlConnection("server=(local)\\NetSDK;Trusted_Connection=yes;database=bdd1");
    SqlCommand mySqlCommand = new SqlCommand("INSERT INTO Clientes
(codigo, nombre, apellido) Values ('1','Pedro', 'Smith')", myConnection);
    SqlCommand mySqlCleanup = new SqlCommand("DELETE FROM Clientes
WHERE codigo = '1'", myConnection);
    try {
      myConnection.Open();
      mySqlCleanup.ExecuteNonQuery();  // remove record that may have been
entered previously.
      mySqlCommand.ExecuteNonQuery();
      Message = "Se ha insertado un registro nuevo en la tabla Clientes";
    }
    catch(Exception e)
    {
      Message= "No se pudo insertar el registro: " + e.ToString();
    }
    finally
    {
      myConnection.Close();
    }
    Console.Write(Message);
  }
}
```

4.4 OBJETOS DataReader

Este objeto es en cierto modo, 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. El siguiente ejemplo, muestra los resultados de una lista de búsqueda utilizando C#.

```
using System;
using System.Data;
using System.Data.SqlClient;

public class ado3
{
  public static void Main()
  {
    ado3 myado = new ado3();
    myado.Run();
  }

  public void Run()
  {
    SqlDataReader myReader = null;
    SqlConnection mySqlConnection = new
SqlConnection("server=(local)\\NetSDK;Trusted_Connection=yes;database=bdd1");
    SqlCommand mySqlCommand = new SqlCommand("select * from clientes",
mySqlConnection);

    try
    {
      mySqlConnection.Open();
      myReader = mySqlCommand.ExecuteReader();
      while (myReader.Read())
      {
        Console.Write(myReader["codigo"].ToString() + "    ");
        Console.WriteLine(myReader["nombre"].ToString());
      }
    }
    catch(Exception e)
    {
      Console.WriteLine(e.ToString());
    }
    finally
    {
      if (myReader != null)  myReader.Close();
      if (mySqlConnection.State == ConnectionState.Open)
        mySqlConnection.Close();
    }
  }
}
```

4.5 OBJETOS DataSet Y DataAdapter

4.5.1 Objetos DataSet

El objeto **DataSet** 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. Sin embargo, 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.

4.5.2 Objetos DataAdapter (OLEDB/SQL)

El objeto **DataAdapter** funciona como un puente entre el objeto **DataSet** y los datos de origen.

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.

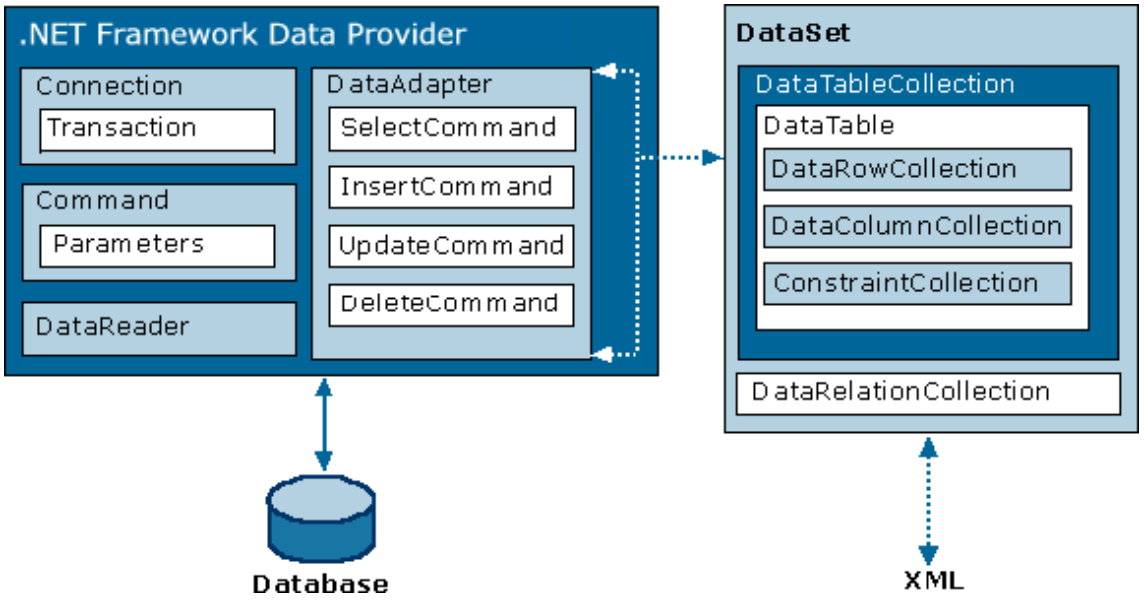


Figura 4.2: Arquitectura de ADO.NET

En el siguiente ejemplo se ilustra la carga de un objeto **DataAdapter** a través de una instrucción *SELECT*. Por último, se devuelven las actualizaciones a la base de datos de origen a través del objeto **DataAdapter**. También se ilustra el uso de varios objetos **DataAdapter** para cargar varias tablas en el objeto **DataSet**, se utiliza el lenguaje C#.

```
using System;
using System.Data;
using System.Data.SqlClient;

public class ado4
{
  public static void Main()
  {
    ado4 myado = new ado4();
    myado.Run();
  }

  public void Run()
  {
    // Create a new Connection and SqlDataAdapter

    SqlConnection myConnection = new
SqlConnection("server=(local)\\NetSDK;Trusted_Connection=yes;database=bdd");
    SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter("Select * from
Region", myConnection);
    SqlParameter workParam = null;

    DataSet myDataSet = new DataSet();

    // Set the MissingSchemaAction property to AddWithKey because Fill will not
cause primary key & unique key information to be retrieved unless AddWithKey is
specified.
    mySqlDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
    mySqlDataAdapter.Fill(myDataSet, "Region");

    DataRow myDataRow1 = myDataSet.Tables["Region"].Rows.Find(2);
    myDataRow1[1] = "Se ha cambiado la descripción de esta región.";

    DataRow myDataRow2 = myDataSet.Tables["Region"].NewRow();
    myDataRow2[0] = 901;
    myDataRow2[1] = "Regin nueva";
    myDataSet.Tables["Region"].Rows.Add(myDataRow2);

    try
    {
      mySqlDataAdapter.Update(myDataSet, "Region");
      Console.Write("El conjunto de datos se actualizó correctamente.");
    }
    catch(Exception e)
    {
      Console.Write(e.ToString());
    }
  }
```

4.6 ADO. NET EN MONO

Con la necesidad de la conexión a bases de datos surge una implementación en Mono conocida como ADO.NET que pretende la comunicación con bases de datos a un alto nivel de abstracción. Permite la manipulación de bases de datos, agregando, insertando, borrando o modificando registros; es muy familiar a la librería de conexión a base de datos ***libgda***[15] de GNOME conocida también como ***gnome-db***, únicamente que integrado con todas las ventajas de .NET al ser un modelo multilenguaje totalmente distribuido.

Para ello, el equipo de personas que trabajan en el proyecto Mono están implementando ADO.NET a partir de muchas de las características de ***libgda*** portando esta biblioteca a C#, consiguiendo la manipulación de bases de datos como ORACLE, PostgreSQL, MySQL, SQLLite entre otras, igualmente con OLEDB entre otros proveedores. Esto permite la conectividad a las bases de datos de una forma sencilla utilizando una clase para la conexión pertinente en la que se le pasa el proveedor, usuario, contraseña y la ubicación de la base de datos con la que queremos conectar. Conforme vayan apareciendo nuevos proveedores se irán implementando nuevas clases en ***libgda*** y aumentará el número de tipos diferentes de bases de datos a las que podremos conectarnos.

Los proveedores de datos, en la terminología de ADO.NET, no son más que simples adaptadores (drivers) que permiten el acceso a un servidor o fuente de datos en concreto.

Microsoft integra el .Net Framework con proveedores de bases de datos para MS SQL Server, Oracle, ODBC, Ole-Db y XML.
Mono integra todos estos proveedores, pero adicionalmente proveedores nativos para MySQL, PostgreSQL e IBM DB2.

[15] Libgda: información en la URL http://www.gnome-db.org/

Las bases de datos no habilitadas directamente pueden ser accedidas a través de *ODBC* u *OLE-DB*. Ambos son implementados sobre sistemas Windows y Unix. Todas esas bases de datos comparten una interfase común, que los hace fácil de usar. También hay un proveedor XML, que habilita a acceder a archivos XML como una base de datos.

4.7 PROVEEDORES DE DATOS PARA ADO.NET EN MONO

Lista de los Proveedores de Datos ADO.NET que trabajan en Mono:

- IBM DB2 Universal Database
- MySQL
- ODBC
- Oracle
- OLE DB
- PostgreSQL
- Microsoft SQL Server
- SQL Lite
- Sybase
- TDS Generic

A continuación tenemos los proveedores que no son soportados por Mono todavía, pero que serán habilitados en el futuro. Actualmente, algunas de estas bases de datos son soportadas vía *ODBC* u *OLE DB*:

- Clases manejadas para *SqlXml*[16], el cual es un proveedor de datos .Net para recuperación de datos XML desde una base de datos Microsoft SQL Server 2000.
- miniSQL [17]
- BerkeleyDB (Sleepycat)[18]

[16] SqlXml: información en : http://msdn.microsoft.com/library/en-us/dnsql2k/html/sqlxml_intromanagedclasses.asp?frame=true
[17] miniSQL: información en: http://www.hughes.com.au/
[18] BerkeleyDB (Sleepycat): información en http://www.sleepycat.com/

- SapDB[19]
- Informix[20]
- Foxpro[21]
- Microsoft Access, que puede ser enlazado, utilizando enlazamientos de C# a herramientas MDB[22]
- Archivos de base de datos tipo *dbase* o *xbase*

A continuación tenemos los proyectos externos a Mono que han creado proveedores ADO.NET y que trabajan sobre Mono:

- *Firebird Interbase* es un proveedor de datos manejado por Firebird SQL. También puede ser usado con bases de datos Interbase. Es escrito totalmente en C# y no requiere una librería cliente. Trabaja en Microsoft .Net y en Mono.
- *Npgsql* es un proveedor de datos manejado por PostgreSQL escrito totalmente en C# y no requiere una librería cliente. Trabaja en Microsoft .Net y en Mono.
- *MySQLNet* es un proveedor de datos manejado por MySQL escrito totalmente en C#. No requiere una librería cliente y trabaja en Microsoft .Net y en Mono.

4.7.1 Proveedor de Datos ADO.NET para la Base de Datos Universal IBM DB2

Fue creado por *Christopher Bockner*[23] y es un proveedor de datos ADO.NET para la base de datos *IBM DB2 Universal Database*[24]. Compila muy bien en Windows y en Linux; trabaja bien en Windows, pero todavía esta bajo pruebas en Linux.

[19] SapDB: información en : http://www.sapdb.org/
[20] Informix: información en: http://www-3.ibm.com/software/data/informix/
[21] Foxpro: información en : http://msdn.microsoft.com/vfoxpro/
[22] MDB tools: información en : http://mdbtools.sourceforge.net/
[23] Christopher Bockner: mail: brianlritchie@hotmail.com
[24] IBM DB2 Universal Database: información en: http://www-3.ibm.com/software/data/db2/

Existe en el namespace **DB2ClientCS** y en el assembly **Mono.Data.DB2Client**. En Mono, el código fuente existe en la ruta: *mcs/class/Mono.Data.DB2Client.*

Requiere una Interface de Nivel de Llamada (*CLI – Call Level Interface*) a la librería compartida IBM DB2. En Windows esta librería es *db2cli.dll*, y este API es muy similar al API ODBC.

Para probarlo necesitamos:

- Tener instalado Mono y funcionando correctamente.
- Tener acceso a una base de datos IBM DB2, si no se lo tiene, se pude conseguir versiones para Windows, Linux y Sun Solaris en *http://www-3.ibm.com/software/data/db2*
- Asegurarse que el assembly *Mono.Data.DB2Client.dll* fue compilado e instalado donde están instaladas las otras librerías de clases.
- Tener una cadena de conexión, también puede ser vía ODBC.

A continuación tenemos un ejemplo de cadena de conexión si contamos con una instalación DSN (*Data Set Name*):

```
"DSN=nombre_de_dsn;UID=mi_usuario;PWD=micontraseña"
```

A continuación tenemos un ejemplo de cadena de conexión si no contamos con un DSN:

```
"DRIVER={DB2Driver};SERVER=localhost;DATABASE=test;UID=myuserid;PASS
WORD=mypassword"
```

Para construir el ejemplo en Unix:

```
mcs DBConnTest.cs -r System.Data.dll -r Mono.Data.DB2Client.dll
```

Para correr la aplicación sobre Mono:

```
mono DBConnTest.exe database userid password
```

A continuación tenemos un ejemplo en C# para utilizar la conexión a la base de datos IBM DB2:

```
using System;
using System.Data;
using Mono.Data.DB2Client;

public class Test
 {
  public static void Main(string[] args)
    {
     string connectionString = "DSN=sample;UID=db2admin;PWD=mypass";
      IDbConnection dbcon = new DB2ClientConnection(connectionString);
      dbcon.Open();
      IDbCommand dbcmd = dbcon.CreateCommand();
      string sql = "CREATE TABLE mono_db2_test1 ( " + "   testid varchar(2),
" + "   testdesc varchar(16) " + ")";
      dbcmd.CommandText = sql;
      dbcmd.ExecuteNonQuery();
      dbcmd.Dispose(); dbcmd = null;
      dbcon.Close(); dbcon = null;
    }
 }
```

4.7.2 Proveedor de Datos ADO.NET para la Base de Datos MySQL

Hay dos proveedores ADO.NET en Mono para la base de datos MySQL:

- Proveedor MySQLNet
- Proveedor Mono.Data.MySql

4.7.2.1 Proveedor MySQLNet

El proveedor MySQLNet, es un proveedor manejado por .Net para MySQL. Es de código abierto, y tiene licencia GPL (*General Public License*). Todavía no se encuentra finalizado, pero los desarrolladores están trabajando activamente en esto.

Para su uso, el namespace se lo utiliza como: **ByteFX.Data.MySQLClient**. Se puede encontrar los instaladores en el URL: *http://sourceforge.net/projects/mysqlnet*

Esta escrito 100% en C# y no requiere una librería cliente. Trabaja sobre Microsoft .Net y sobre Mono, es decir que compila y corre sobre estas dos plataformas. El soporte transaccional está implementado (no todos los tipos de tablas soportan esto).

Para probar este proveedor de datos se necesita lo siguiente:

- Tener instalada a la base de datos MySQL, se puede encontrar instaladores en *http://www.mysql.com/downloads/index.html*
- Instalar MySQLNet. El assembly binario *ByteFX.Data.dll* necesita estar instalado en el mismo lugar que las librerías de clase de Mono.
- MySQL Net requiere *SharpZipLib*[25], el cual es una librería Zip escrita 100% en C#. Es usada para compresión y descompresión de datos enviados y recibidos a través de la red. El assembly binario *SharpZipLib.dll* debe ser instalado en el mismo lugar que las librerías de clases de Mono.
- Tener una cadena de conexión, la cual tiene el formato:

```
"Server=hostname;" + "Database=database;" + "User ID=username;" +
"Password=password"
```

A continuación tenemos un ejemplo utilizando C#, el cual debemos guardarlo

```
using System;
using System.Data;
using ByteFX.Data.MySqlClient;
 public class Test
 {
  public static void Main(string[] args)
    {
     string connectionString = "Server=localhost; Database=test; User ID=myuserid;
Password=mypassword;";
      IDbConnection dbcon;
      dbcon = new MySqlConnection(connectionString);
      dbcon.Open();
      IDbCommand dbcmd = dbcon.CreateCommand();
       string sql = "SELECT nombre, apellido FROM empleado";
       dbcmd.CommandText = sql;
       IDataReader reader = dbcmd.ExecuteReader();
```

[25] *SharpZipLib* : información en: http://www.icsharpcode.net/OpenSource/SharpZipLib/Default.aspx

```
        while(reader.Read()) {
            string PrimerNombre = (string) reader["nombre"];
            string Apellido = (string) reader["apellido"];
            Console.WriteLine("Nombre: " + PrimerNombre + " " + Apellido);
        }
        // clean up
        reader.Close(); reader = null;
        dbcmd.Dispose(); dbcmd = null;
        dbcon.Close(); dbcon = null;
    }
}
```

El comando para compilar el archivo es:

```
mcs EjemploByteFX.cs -r System.Data.dll \ -r ByteFX.Data.dll
```

Para correr el ejemplo escribimos:

```
mono EjemploByteFX.exe
```

4.7.2.2 Proveedor Mono.Data.MySql

Fue iniciado por *Daniel Morgan*[26] utilizando enlazamientos desde C# hasta MySQL. Esta escrito en C# y usa la librería *MySQL C Client*[27]. Existe en el namespace **Mono.Data.MySql** y en el assembly **Mono.Data.MySql**. Trabaja en Windows y en Linux a través de la librería compartida *MySQL client* (*libmySQL.dll* en Windows y *libmysqlclient.so* en Linux)

Podemos usar el ejemplo anterior para probarlo, lo único que tenemos que cambiar es la línea:

```
using ByteFX.Data.MySqlClient;
```

, por esta otra:

```
using Mono.Data.MySql;
```

Para construir el ejemplo anterior, debemos guardarlo como un archivo de extensión *.cs*.

[26] Daniel Morgan: email: justdandy43@hotmail.com
[27] MySQL C Client: información en: http://www.mysql.de/doc/en/C.html

El comando para compilar el archivo es:

```
mcs EjemploMySqlClient.cs -r System.Data.dll \ -r Mono.Data.MySql.dll
```

Para correr el ejemplo escribimos:

```
mono EjemploMySqlClient.exe
```

4.7.3 Proveedor de Datos ADO.NET para ODBC

Fue creado por *Brian Ritchie*[28], y es un proveedor ADO.NET para orígenes de datos que tiene soporte a ODBC.

Existe en el namespace **System.Data.Odbc** y en el assembly **System.Data***;* trabaja en Windows y el Linux. En Windows trabaja utilizando el archivo nativo de Windows: *odbc32.dll*.

En Linux trabaja de dos maneras:

- Utilizando *unixODBC*[29] , el cual tiene soporte comercial con la empresa *Easysoft*[30]*.* Para esto utiliza la librería *libodbc.so*
- Utilizando *iODBC*[31], el cual tiene soporte comercial con la empresa: *OpenLink Software*[32]*.* Para esto utiliza la librería *libiodbc.so*

ODBC es muy útil, ya que se puede conectar a varias bases de datos que tengan un manejador ODBC instalado, las bases de datos que incorporan este manejador son: MySQL, PostegreSQL, Oracle, Interbase, Sybase, IBM DB2 Universal DataBase, MS Access (utilizando herramientas *MDB*[33]).

[28] Bryan Ritchie. Información en: http://www12.brinkster.com/brianr/ Email: brianlritchie@hotmail.com
[29] *unixODBC* : información en : http://www.unixodbc.org/
[30] *Easysoft* : información en: http://www.easysoft.com/
[31] *iODBC: información en:* http://www.iodbc.org/
[32] *OpenLink Software* : información en : http://oplweb.openlinksw.com:8080/download/
[33] MDB: información en: http://mdbtools.sourceforge.net/

Como ejemplo, para acceder al proveedor ODBC para Mono utilizamos el namespace *System.Data.Odbc,* al cual lo vamos a utilizar con el manejador ODBC de MySQL llamado *MyODBC.*

Si contamos con una instalación DSN podemos utilizar una cadena de conexión de la siguiente forma:

```
"DSN=dataSetName;UID=myuserid;PWD=mypassword"
```

Si no contamos con una instalación DSN, podemos utilizar una cadena de conexión de la siguiente forma:

```
"DRIVER={MySQL ODBC 3.51 Driver};" + SERVER=localhost;DATABASE=test;" +
"UID=myuserid;PASSWORD=mypassword;" + "OPTION=3";
```

A continuación tenemos un ejemplo utilizando C#, la cual necesita una instalación ODBC con DSN llamada *MYSQLDSN* que accede a una base de datos MySQL a través del manejador MyODBC. El nombre del host es *localhost* y el nombre de la base de datos es *test.*
Requiere de una tabla que debe ser creada llamada *empleado* con las columnas *nombre* y *apellido,* las cuales son de tipo *varchar*.

```
using System;
using System.Data;
using System.Data.Odbc;

public class Test
{
   public static void Main(string[] args)
   {
     string connectionString = "DSN=MYSQLDSN; UID=myuserid;
PWD=mypassword";
      IDbConnection dbcon;
      dbcon.Open();
      dbcon = new OdbcConnection(connectionString);
      IDbCommand dbcmd = dbcon.CreateCommand();
      string sql = "SELECT nombre, apellido FROM empleado";
      dbcmd.CommandText = sql;
      IDataReader reader = dbcmd.ExecuteReader();
```

```
        while(reader.Read()) {
            string FirstName = reader["nombre"];
            string LastName = reader["apellido"];
            Console.WriteLine("Name: " + FirstName + " " + LastName);
        }
        // limpiar la memoria.
        reader.Close(); reader = null;
        dbcmd.Dispose();dbcmd = null;
        dbcon.Close();dbcon = null;
    }
}
```

Para construir el ejemplo anterior, debemos guardarlo como un archivo de extensión *.cs*. El comando para compilar el archivo es:

```
mcs EjemploOdbc.cs -r System.Data.dll
```

Para correr el ejemplo escribimos:

```
mono EjemploOdbc.exe
```

4.7.4 Proveedor de datos para PostgreSQL

PostgreSQL fue el primer proveedor ADO.NET creado en Mono. Existen dos proveedores de datos ADO.NET para PostgreSQL en Mono:

- Npgsql
- Mono.Data.PostgreSQL

4.7.4.1 Proveedor Npgsql

Fue creado por *Francisco Figueiredo Jr.* y tiene muchos desarrolladores trabajando en desarrollar esta herramienta. Es un proveedor de datos administrado por .NET para PostgreSQL, el cual es escrito 100% en C# y no requiere una librería cliente. Trabaja tanto en Mono como en Microsoft.NET.

El namespace **Npgsqly** el assembly **Npgsql** se lo puede encontrar en *mcs* en la ruta: *mcs/class/Npgsql*

Permite soporte para reportes históricos. Para usarlo es necesario ubicar código como el siguiente dentro del programa en el que se trabaja:

```
// Enable logging.
NpgsqlEventLog.Level = LogLevel.Debug;              // LogLevel.
NpgsqlEventLog.LogName = "NpgsqlTests.LogFile";   // LogFile.
```

Npgsql ha reemplazado a **Mono.Data.PostgreSqlClient** como el proveedor con opción a ser usado. Sin embargo, **Mono.Data.PostgreSqlClient** va a permanecer en estado de depreciación hasta que nadie lo use más, entonces podrá ser removido.

Para probar este proveedor de datos es necesario tener funcionando correctamente Mono y mcs. Debemos obtener *Npgsql*[34], y asegurarse de que el assembly binario *Npgsql.dll* está instalado en el mismo lugar donde las librerías de clases de mono estén ubicadas. Aunque, en las últimas versiones de mono, viene incluido este paquete como parte de la distribución.

Es necesario tener acceso local o remoto a una base de datos PostgreSQL. Se puede usar PostgreSQL porque es un software libre, tiene una librería cliente que es fácil de usar, es fácil de instalar, no es difícil la configuración después de la instalación y corre bajo: Linux, Windows, Unix y otras plataformas. Además nos permite utilizar la funcionalidad de *System.Data* en Mono mucho más rápido.

Las pruebas utilizando *System.Data* usan una conexión para conectarse a PostgreSQL de la siguiente manera:

```
"Server=localhost;Database=test;User ID=postgres;Password=fun2db"
    (or)
"host=localhost;dbname=test;user=postgres;password=fun2db"
```

A continuación tenemos un ejemplo en C# para Npgsql:

[34] Npgsql: información en: http://gborg.postgresql.org/project/npgsql/projdisplay.php

```
using System;
using System.Data;
using Npgsql;
public class Test
 {
   public static void Main(string[] args)
    {
     string connectionString ="Server=localhost; Database=test; User
    ID=postgres; Password=fun2db;";
     IDbConnection dbcon;
     dbcon.Open();
     dbcon = new NpgsqlConnection(connectionString);
     IDbCommand dbcmd = dbcon.CreateCommand();
    string sql = "SELECT nombre, apellido FROM empleado";
    dbcmd.CommandText = sql;
    IDataReader reader = dbcmd.ExecuteReader();
     while(reader.Read()) {
              string FirstName = reader["nombre"];
              string LastName = reader["apellido"];
              Console.WriteLine("Name: " + FirstName + " " + LastName);
      }
     // clean up
     reader.Close(); reader = null;
     dbcmd.Dispose(); dbcmd = null;
     dbcon.Close(); dbcon = null;
   }
 }
```

Es necesario guardar este ejemplo como un archivo de C#, por ejemplo *EjemploNpgsql.cs*. Para compilar este ejemplo en Linux:

```
mcs EjemploNpgsql.cs -r System.Data.dll \  -r Npgsql.dll
```

Para correr el ejecutable:

```
mono EjemploNpgsql.exe
```

4.7.4.2 Mono.Data.PostgreSQL

Es un proveedor de datos Mono para el sistema de administración de base de datos cliente/servidor PostgreSQL. Esta escrito en C# y tiene ligamientos C# a la librería cliente de PostgreSQL *pq.dll* en Windows y en *libpq.so* en Linux.

Existe en el namespace **Mono.Data.PostgreSql** y en el assembly **Mono.Data.PostgreSql**.

A continuación tenemos un ejemplo en C# utilizando este proveedor:

```
using System;
 using System.Data;
 using Mono.Data.PostgreSqlClient;

 public class Test
 {
    public static void Main(string[] args)
    {
       string connectionString =
          "Server=localhost;" + "Database=test;" + "User ID=postgres;" +
          "Password=fun2db;";
       IDbConnection dbcon;
       dbcon = new PgConnection(connectionString);
       dbcon.Open();
       IDbCommand dbcmd = dbcon.CreateCommand();
       string sql = "SELECT nombre, apellido FROM empleado";
       dbcmd.CommandText = sql;
       IDataReader reader = dbcmd.ExecuteReader();
       while(reader.Read()) {
           string Nombre = reader["nombre"];
           string Apellido = reader["apellido"];
           Console.WriteLine("Nombres: " + Nombre + " " + Apellido);
       }
       reader.Close(); reader = null;
       dbcmd.Dispose(); dbcmd = null;
       dbcon.Close(); dbcon = null;
    }
 }
```

Guardamos el ejemplo como *TestExample.cs*. Para compilarlo utilizamos:

```
mcs EjemploPgsql.cs -r System.Data.dll \
        -r Mono.Data.PostgreSqlClient.dll
```

Para correrlo ejecutamos:

```
mono EjemploPgsql.exe
```

4.7.5 Proveedor de Datos para Oracle

Fue creado por *Daniel Morgan* y *Tim Coleman*[35], y es un proveedor de datos ADO.NET para las bases de datos Oracle. Trabaja tanto en Linux como en Windows y se han hecho pruebas para Oracle 8i, donde trabaja perfectamente; para Oracle 9i no se lo ha probado, por tanto hay la posibilidad de que funcione también con esta distribución.

Usa el *Oracle CLI* (*Call Level Interface*), la cual es una librería de C (API) para el software cliente de Oracle. Internamente, el proveedor *OracleClient*, tiene un *OCI* (*Oracle Call-level Interface*) abstraído al modelo de programación orientado a objetos.

Existe en el namespace **System.Data.OracleClient** y en el assembly **System.Data.OracleClient**.

Puede tener múltiples conexiones con diferentes transacciones, donde cada transacción esta separada de otras, de esta manera, un *rollback* o un *commit* en una transacción no afecta a otras.

Para probarlo, es necesario tener funcionando mono y mcs, tener acceso a la base de datos Oracle 8i o 9i. Si no se lo tiene se lo puede descargar desde la url: *http://www.oracle.com*. En caso de que queramos conectarnos remotamente a una base de datos Oracle, necesitaremos el software cliente de Oracle.

El registro a la *Oracle Technology Network (Red de Tecnología de Oracle)* es gratuito en la url: *http://technet.oracle.com* . Si se instala en Linux, es recomendable buscar en la red como otros lo han instalado.

Debemos asegurarnos de que el assembly *System.Data.OracleClient.dll* esta construido, si no lo esta, ubicarnos donde se encuentre la distribución de *System.Data.OracleClient* y hacer:

[35] Tim Coleman: email: tim@timcoleman.com

```
make -f makefile.gnu
```

Por ultimo debemos tener una cadena de conexión con el formato:

```
"Data Source=tnsname;User ID=userid;Password=password"
```

A continuación tenemos un ejemplo en C#:

```
using System;
using System.Data;
using System.Data.OracleClient;
public class Test
 {
  public static void Main (string[] args)
   {
    string connectionString = "Data Source=testdb;" +
            "User ID=scott;" + "Password=tiger;";
    OracleConnection dbcon = null;
    dbcon = new OracleConnection (connectionString);
    dbcon.Open ();
    OracleCommand dbcmd = dbcon.CreateCommand ();
    string sql = "SELECT nombre, trabajo FROM scott.emp";
    dbcmd.CommandText = sql;
    OracleDataReader reader = dbcmd.ExecuteReader ();
    while (reader.Read ()) {
      string NombreEmpleado = (string) reader["nombre"];
      string Trabajo = (string) reader["trabajo"];
        Console.WriteLine ("Nombre del empleado: {0}  Trabajo: {1}",
        NombreEmpleado, Trabajo);
     }
     // clean up
     reader.Close (); reader = null;
     dbcmd.CommandText = sql;
     dbcmd.ExecuteNonQuery ();
     dbcmd.Dispose (); dbcmd = null;
     dbcon.Close (); dbcon = null;
  }
}
```

Guardamos el archivo con formato C#, por ejemplo: *EjemploOracle.cs*

Para compilar el archivo utilizamos:

```
mcs EjemploOracle.cs -r System.Data.dll \  -r System.Data.OracleClient.dll
```

Para ejecutarlo utilizamos:

```
Mono EjemploOracle.exe
```

4.7.6 Proveedor de datos para Microsoft SQL Server

Fue creado por *Tim Coleman*, y es un proveedor de datos ADO.NET para la base de datos Microsoft SQL Server 7 o 2000, el cual está implementado 100% en C#.

Existe en el namespace **System.Data.SqlClient** y en el assembly **System.Data**.

Usa el Protocolo TDS versión 7.0, para esto requiere el assembly **Mono.Data.Tds.dll**, el cual implementa el protocolo TDS en lenguaje C# al 100%. Se utilizó como recursos a los proyectos *FreeTDS*[36] y *jTDS*[37]. Por utilizar este protocolo, es similar a los proveedores **Mono.Data.TdsClient** y **Mono.Data.SybaseClient**.

Para probar este proveedor de datos debemos tener instalado y funcionando tanto mono como mcs. Debemos tener acceso a la base de datos Microsoft SQL Server o podemos descargarlo desde: *http://www.microsoft.com/sql/default.asp*.

Si usamos Microsoft SQL Server 2000, debemos asegurarnos que por lo menos tenemos el *Service Pack 3* para Microsoft SQL 2000. Si se está usando MSDE 2000 debemos asegurarnos de tener el *Service Pack 3* especial para MSDE 2000.

La versión *SqlClient* de Mono no soporta conexiones confiables ni seguridad integrad. Se necesita explícitamente usar un identificador de usuario y una contraseña autenticada por SQL Server.

Se necesita tener una cadena de conexión con el siguiente formato:

```
Server=hostname;Database=databaseName;User ID=userid;Password=password
```

[36] FreeTDS: información en http://www.freetds.org
[37] jTDS: información en http://jtds.sourceforge.net

El parámetro *Server* puede ser usada con tres alternativas:

Definición del servidor	*Ejemplo*
hostname	Server = Mi-host
hostname, puerto	Server = Mi-host,1433
hostname\\instancia	Server = Mi-host\\NETSDK

A continuación un ejemplo en C#

```
using System;
using System.Data;
using System.Data.SqlClient;
public class Test
 {
     public static void Main(string[] args)
      {
       string connectionString = "Server=localhost;" +  "Database=pubs;" +
       "User ID=myuserid;" +  "Password=mypassword;";
       IDbConnection dbcon;
       dbcon = new SqlConnection(connectionString); dbcon.Open();
       IDbCommand dbcmd = dbcon.CreateCommand();
       string sql = "SELECT nombre, apellido FROM empleado";
       dbcmd.CommandText = sql;
       IDataReader reader = dbcmd.ExecuteReader();
       while(reader.Read()) {
              string N = (string) reader["nombre"];
             string A = (string) reader["apellido"];
             Console.WriteLine("Nombre: " + N + " " + A);
          }
       reader.Close(); reader = null;
       dbcmd.Dispose(); dbcmd = null;
       dbcon.Close(); dbcon = null;
      }
}
```

Guardamos el ejemplo como un archivo de C#, por ejemplo EjemploMsql.cs

Para compilarlo ponemos:

```
mcs EjemploMsql.cs -r System.Data.dll
```

Para ejecutar el ejemplo ponemos:

```
mono EjemploMsql.exe
```

4.7.7 Proveedor de datos para Sybase

Fue creado por *Tim Coleman*, y es un proveedor de datos ADO.NET par la base de datos Sybase SQL Server. Está implementado 100% en C#.

Usa el Protocolo TDS versión 5.0, para esto requiere el assembly **Mono.Data.Tds.dll***,* el cual implementa el protocolo TDS en lenguaje C# al 100%. Se utilizó como recursos a los proyectos *FreeTDS* y *jTDS*. Debido a que utiliza este protocolo, es similar a los proveedores *Mono.Data.TdsClient* y *Mono.Data.SqlClient.*

Para poder probarlo debemos tener instalado mono y mcs, y tener acceso a la base de datos Sybase o descargarla de: *http://www.sybase.com/downloads*

Debemos tener una cadena de conexión con el formato:

```
Server=hostname;Database=databaseName;User ID=userid;Password=password
```

El parámetro *Server* puede ser usado de dos maneras:

Definición del servidor	*Ejemplo*
Hostname	Server = Mi-host
hostname, port	Server = Mi-host,1533

A continuación tenemos un ejemplo utilizando C#:

```
using System;
using System.Data;
using Mono.Data.SybaseClient;
public class Test
 {
  public static void Main(string[] args)
   {
    string connectionString = "Server=localhost; Database=pubs;" +
        "User ID=myuserid; Password=mypassword;";
    IDbConnection dbcon;
    dbcon = new SybaseConnection(connectionString);
    dbcon.Open();
    IDbCommand dbcmd = dbcon.CreateCommand();
    string sql = "SELECT nombre, apellido FROM empleado";
    dbcmd.CommandText = sql;
    IDataReader reader = dbcmd.ExecuteReader();
```

```
      while(reader.Read()) {
        string N = (string) reader["nombre"];
        string A = (string) reader["apellido"];
        Console.WriteLine("Nombre: " + N + " " + A);
       }
       // clean up
       reader.Close(); reader = null;
       dbcmd.Dispose(); dbcmd = null;
       dbcon.Close(); dbcon = null;
    }
}
```

Para compilar este ejemplo, lo guardamos con extensión de C# y ponemos:

```
mcs EjemploSybase.cs -r System.Data.dll \ -r Mono.Data.SybaseClient.dll
```

Para ejecutar el ejemplo:

```
mono EjemploSybase.exe
```

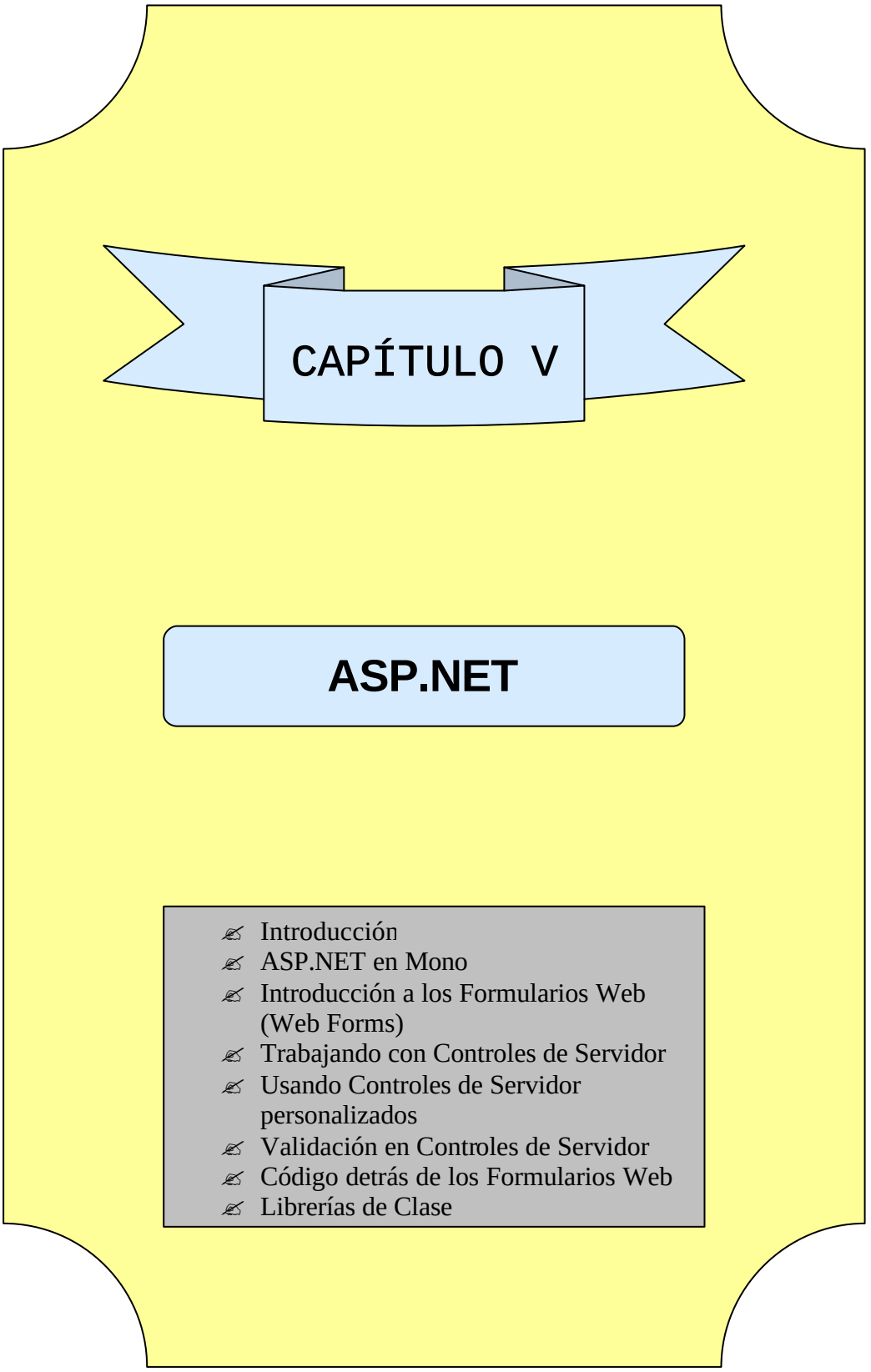
Bibliografía

Libros

- DUBOIS, Paul, “MySQL”, Editorial Prantice Hall,2001

Referencias WWW

- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconadonetarchitecture.asp
- http://www.dnjonline.com/articles/essentials/iss22_essentials.html
- http://www.monografias.com/trabajos14/informe-ado-net/informe-ado-net.shtml
- http://msdn.microsoft.com/msdnmag/issues/1100/adoplus/default.aspx
- http://es.gotdotnet.com/quickstart/howto/doc/adoplus/ADOPlusOverview.aspx
- http://www.mysql.com/articles/dotnet/


CAPÍTULO V
ASP.NET
Introducción
ASP.NET en Mono
Introducción a los Formularios Web (Web Forms)
Trabajando con Controles de Servidor
Usando Controles de Servidor personalizados
Validación en Controles de Servidor
Código detrás de los Formularios Web
Librerías de Clase

5.1 INTRODUCCIÓN

ASP.NET es una estructura de programación construida sobre el CLR (*Common Language Runtime*) que puede ser usado en un servidor para construir poderosas aplicaciones Web. ASP.NET ofrece algunas ventajas importantes sobre los modelos de desarrollo Web previos:

- **Rendimiento mejorado:** ASP.Net esta compilado en código CLR (*Common Language Runtime*) corriendo en el servidor. Diferente a sus intérpretes predecesores, ASP.NET puede tomar ventaja en ligamiento rápido, compilación *just in time* (justo a tiempo), optimización nativa, etc. Esto aumenta mucho el rendimiento.

- **Poder y Flexibilidad:** Debido a que ASP.NET está basada en el CLR, el poder y la flexibilidad de la plataforma entera esta habilitada a los desarrolladores de aplicaciones Web. La librería de clases y las soluciones para acceso a datos es mucha más accesible desde la Web. ASP.NET también es independiente del lenguaje, con esto podemos escoger el lenguaje que mejor se adapte a nuestra aplicación o parte de ella. Adicionalmente, la interoperabilidad del CLR (*Common Language Runtime*) garantiza que el empleo de desarrollo basado en COM sea preservado cuando se migra hacia ASP.NET.

- **Simplicidad:** ASP.NET hace que sea fácil organizar tareas comunes, desde una simple autentificación de clientes hasta la configuración de un sitio. Por ejemplo, en una página ASP.NET se puede construir interfaces donde esté claramente separada la lógica de la aplicación: el código de presentación y la utilización de eventos. Adicionalmente, el CLR simplifica el desarrollo con servicios de código manejado como puede ser el conteo de referencia automático y la recolección de basura.

- **Manejabilidad:** ASP.NET emplea un sistema de configuración jerárquico basado en texto, con lo cual se simplifica la aplicación de configuraciones al entorno del servidor y a las aplicaciones Web. Debido a que la información de la configuración se almacena en un archivo de texto plano, los nuevos cambios en la configuración pueden ser aplicados sin la ayuda de herramientas de administración locales. Una aplicación ASP.NET se despliega a un servidor simplemente copiando los archivos necesarios al servidor. No es necesario reiniciar el servidor, ni siquiera al remplazar código compilado y que esta corriendo.

- **Escalabilidad y Disponibilidad:** ASP.NET ha sido designado con escalabilidad en mente, con características específicamente hechos para improvisar rendimiento en ambientes agrupados y multiprocesadores. Además, los procesos son cerradamente monitoreados y manejados por el compilador de ASP.NET, así que si algo se conduce mal (bloqueos, escapes), un nuevo proceso puede ser creado en este lugar para ayudar a que la aplicación este constantemente disponible a cualquier requerimiento.

- **Personalización y Extensibilidad:** ASP.NET entrega una arquitectura bien fabricada que permite a los desarrolladores poner su propio código en el nivel adecuado. De hecho, es posible extender o reemplazar cualquier subcomponente del compilador de ASP.NET con su propio componente reescrito.

- **Seguridad:** Con una adecuada configuración por cada aplicación. Nos podemos asegurar que la aplicación sea muy segura.

5.2 ASP.NET EN MONO

El soporte de ASP.NET en Mono esta dividido en dos partes, los cuales son funcionales hasta el momento:

- Formularios Web (Infraestructura de aplicaciones Web)
- Servicios Web (el sistema basado en SOAP - *Single Object Access Protocol*)

En mono, ASP.NET trabaja tanto utilizando el servidor Web *XSP* que se incluye en la distribución de mono, así como también utilizando un módulo para el servidor Apache llamado *mod_mono*

5.2.1 Hosting

ASP.NET provee una interfase para hosting. Actualmente, soporta dos mecanismos de hosting:

- XSP: Es un servidor web simple escrito en C# el cual es muy liviano y puede ser usado para correr aplicaciones ASP.NET. Esta integrado en las clases **System.Web,** la mayoría están bajo el *namespace* **System.Web.Compilation**
- mod_mono: Es un módulo de Apache que puede almacenar el compilador de Mono y el compilador de ASP.NET. Trabaja con las versiones 1.3 y 2.0 de Apache.

5.3 INTRODUCCIÓN A LOS FORMULARIOS WEB (*WEB FORMS*)

Los Formularios Web de ASP.NET es un modelo de programación escalable del CLR que puede ser usado en un servidor para generar dinámicamente páginas Web.

Es entendido como una evolución lógica de ASP (*Active Server Pages*), los Formularios Web ASP.NET han sido específicamente diseñados para

direccionar un número de deficiencias claves en el modelo previo. En particular, provee:

- La habilidad de crear y usar controles de interfaz de usuario reusables que pueden ser encapsulados en una funcionalidad común y de este modo reducir el aumento de código que un desarrollador de páginas Web tiene que escribir.
- La habilidad para los desarrolladores de estructurar claramente la lógica de la página de un modo ordenado.

5.3.1 Iniciando con Formas Web

Las páginas con Formularios Web ASP.NET son archivos de texto con una extensión ***.aspx*** . Cuando un browser cliente (Internet Explorer, Navigator, etc.) requiere una página aspx, el compilador de ASP.NET compila el archivo pedido dentro de una de las clases .NET. Esta clase puede entonces ser usada para procesar dinámicamente los requerimientos. Note que el archivo ***.aspx*** es compilado únicamente la primera vez que es accedido; la instancia compilada es entonces rehusada a través de múltiples requerimientos.

Una página ASP.NET puede ser creada simplemente tomando un archivo HTML ya existente y cambiarlo de nombre a la extensión ***.aspx*** (no se requiere modificar el código).

El siguiente ejemplo nos muestra un código simple escrito en HTML.

```
<html>
  <body>
     <center>
     <form action="intro1.aspx" method="post">
        <h3> Nombre: <input id="Nombre" type=text>
        Category:  <select id="Categoria" size=1>
                        <option>Cultura</option>
                        <option>Negocios</option>
                    </select>
        <input type=submit value="Busqueda">
        </h3>
     </form>
     </center>
  </body>
</html>
```

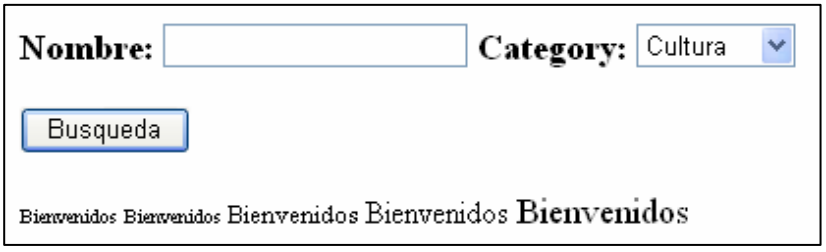
5.3.2 Usando los bloques ASP: <% y %>

ASP.NET provee compatibilidad sintáctica con páginas ASP existentes. Esto incluye soporte para bloques de código **<% %>** que pueden ser entremezclados con contenido HTML en un archivo **.aspx** .

A diferencia de ASP, el código usado entre los bloques **<% %>** es actualmente compilado, no interpretado. Esto da como resultado un gran rendimiento en tiempo de ejecución.

El siguiente ejemplo demuestra como estos bloques pueden ser usados para hacer un ciclo en un bloque HTML, en este caso para incrementar un poco más el tamaño del texto cada vez.

```
<%@ Page Language="C#"%>
<form action="intro2.aspx" method="post">
        <h3> Nombre: <input id="Nombre" type=text>
        Category:  <select id="Categoria" size=1>
                <option>Cultura</option>
                <option>Negocios</option>
                <option>Arte</option>
              </select>
        </h3>
        <input type=submit value="Busqueda">
        <p>
        <% for (int i=0; i <7; i++) { %>
           <font size="<%=i%>"> Bienvenidos </font>
        <% }%>
 </form>
```



5.4 TRABAJANDO CON CONTROLES DE SERVIDOR

Es indispensable conocer algunos conceptos centrales y acciones comunes realizadas cuando se utilizan los controles de servidor ASP.NET en una página.

En adición, el uso de los bloques de código **<% %>** para programar contenido dinámico, los desarrolladores pueden usar controles de servidor ASP.NET para programar páginas Web. Los controles de servidor son declarados dentro del archivo **.aspx** usando etiquetas personalizadas o etiquetas intrínsecas HTML que contienen un atributo con el valor **runat="server"**.

Las etiquetas intrínsecas HTML son tomadas por uno de los controles en el namespace **System.Web.UI.HtmlControls**. Cualquier etiqueta que no mapea a uno de los controles es asignado al tipo **System.Web.UI.HtmlControls.HtmlGenericControl**.

Estos controles de servidor automáticamente mantienen cualquier valor ingresado por el cliente al servidor. Este estado del control no es almacenado en el servidor, en lugar de ello se almacena dentro de un campo de la forma de tipo **<input type="hidden">** que se crea en los requerimientos. Además no es necesario un script en el lado del cliente.

Existen dos tipos de controles de servidor:

- System.Web.UI.HtmlControls
- System.Web.UI.WebControls

5.4.1 CONTROLES System.Web.UI.Html

Los controles de servidor HTML son elementos HTML expuestos al servidor para poder programar sobre ellos. Estos controles exponen un modelo de objetos que mapean muy cerradamente los elementos HTML a los cuales traducen.

A continuación presentamos una lista de estos controles.

HtmlAnchor	HtmlButton	HtmlForm	HtmlGenericControl
HtmlImage	HtmlInputButton (Button)	HtmlInputButton (Reset)	HtmlInputButton (Submit)
HtmlInputCheckBox	HtmlInputFile	HtmlInputHidden	HtmlInputImage
HtmlInputRadioButton	HtmlInputText (Password)	HtmlInputText (Text)	HtmlSelect
HtmlTable	HtmlTableCell	HtmlTableRow	HtmlTextArea

5.4.2 CONTROLES System.Web.UI.Web

Los controles de servidor Web son controles ASP.NET con un modelo de objetos abstracto y fuertemente ejemplar. Estos controles incluyen no únicamente controles de tipo formulario como botones y cajas de texto, sino que también incluyen controles de propósito especial como un calendario. Estos controles son más abstractos que los controles de servidor HTML, en aquellos, su modelo de objetos no es necesariamente reflejado en la sintaxis HTML.

A continuación presentamos una lista de estos controles.

AdRotator	Button	Calendar	CheckBox
CheckBoxList	CompareValidator	CustomValidator	DataGrid
DataList	DropDownList	HyperLink	Image
ImageButton	Label	LinkButton	ListBox
Panel	PlaceHolder	PlaceHolder	RadioButtonList
RangeValidator	RegularExpressionValidator	Repeater	RequiredFieldValidator
Table	TableCell	TableRow	TextBox
ValidationSummary	XML		

5.4.2.1 Control AdRotator

El control **AdRotator** presenta imágenes de aviso que cuando son presionadas, se navega a una nueva locación Web. Cada vez la página es cargada en el browser, y el aviso es randomicamente seleccionado de una lista predefinida.

El siguiente ejemplo ilustra el uso de este control.

```
<asp:AdRotator id="ar1" AdvertisementFile="Ads.xml" BorderWidth="1"
runat=server />
```

La rotación anexa un aviso que esta definido en un archivo XML. El archivo XML es el siguiente:

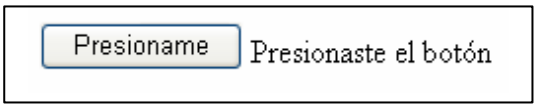
```
<Advertisements>
  <Ad>
    <ImageUrl>/quickstart/aspplus/images/banner1.gif</ImageUrl>
    <NavigateUrl>http://www.go-mono.com</NavigateUrl>
    <AlternateText>Microsoft.com</AlternateText>
    <Keyword>Computers</Keyword>
    <Impressions>80</Impressions>
  </Ad>
</Advertisements>
```

5.4.2.2 Control Button

El control **Button** provee un control comando de estilo botón que es usado para enviar una página con formas Web al servidor.

```
<html>
<head>
  <script language="C#" runat="server">
    void Button1_Click(Object Sender, EventArgs e) {
      Label1.Text="Presionaste el botón";
    }
  </script>
</head>
<body>
 <form runat=server>
  <asp:Button id=Button1 Text="Presioname" onclick="Button1_Click"
runat="server" />
  </form>
</body>
</html>
```

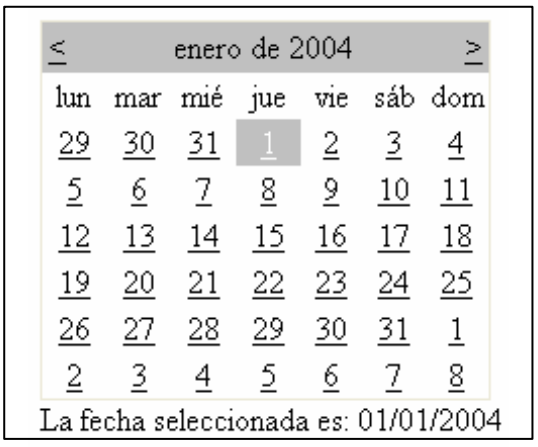
El ejemplo anterior muestra el uso de un botón con su respectivo evento escrito en C#. A continuación, la respectiva salida a pantalla:



5.4.2.3 Control Calendar

El control **Calendar** muestra un calendario mensual, en el cual el usuario puede seleccionar fechas. El siguiente ejemplo muestra como usarlo:

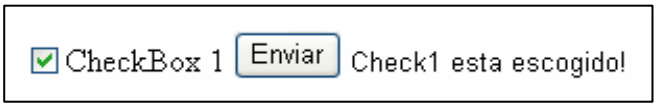
```
<html>
<head>
   <script language="C#" runat="server">
      void Date_Selected(object s, EventArgs e) {
         Label1.Text = "La fecha seleccionada es: " +
Calendar1.SelectedDate.ToShortDateString();
      }
   </script>
</head>
<body>
  <form runat=server>
    <asp:Calendar id=Calendar1 onselectionchanged="Date_Selected"
runat="server" />
    <asp:Label id=Label1 runat="server" />
   </form>
</body>
</html>
```



5.4.2.4 Control CheckBox

Este control acepta entrada de tipo **Boolean** (verdadero o falso). Cuando se selecciona, la propiedad **Checked** es verdadera. El siguiente ejemplo muestra su uso:

```
<html>
<head>
   <script language="C#" runat="server">
      void SubmitBtn_Click(Object Sender, EventArgs e) {
        if (Check1.Checked == true) {
           Label1.Text = "Check1 esta escogido!";
         }
      }
   </script>
</head>
<body>
   <form runat=server>
      <asp:CheckBox id=Check1 Text="CheckBox 1" runat="server" />
      <asp:button text="Enviar" OnClick="SubmitBtn_Click" runat=server/>
      <asp:Label id=Label1 font-name="arial" font-size="10pt"
runat="server"/>
  </form>
</body>
</html>
```



5.4.2.5 Control CheckBoxList

El control **CheckBoxList** provee una lista revisable de múltiple selección. Como otros controles de lista, este tiene una colección de ítems con miembros que corresponden a cada ítem en la lista. A continuación un ejemplo:

```
<asp:CheckBoxList id=Check1 runat="server">
          <asp:ListItem>Item 1</asp:ListItem>
          <asp:ListItem>Item 2</asp:ListItem>
          <asp:ListItem>Item 3</asp:ListItem>
 </asp:CheckBoxList>
```

5.4.2.6 Control DataGrid

El control **DataGrid** muestra datos tabulados y opcionalmente soporta selección, ordenamiento, paginación y edición de los datos. Por defecto, **DataGrid** genera un salto de columna para cada campo en el origen de datos (con la propiedad **AutoGenerateColumns=true**). Cada campo se ubica en una columna separada, en el orden en que vienen los datos. Los nombres de los campos aparecen en la cabecera de cada columna de la grilla.

```
<%@ Import Namespace="System.Data" %>
<html>
<script language="C#" runat="server">
   ICollection CreateDataSource() {
      DataTable dt = new DataTable();
      DataRow dr;
      dt.Columns.Add(new DataColumn("Value Entero", typeof(Int32)));
      dt.Columns.Add(new DataColumn("Valor String", typeof(string)));
      for (int i = 0; i < 3; i++) {
         dr = dt.NewRow();
         dr[0] = i;
         dr[1] = "Item " + i.ToString();
         dt.Rows.Add(dr);
      }
      DataView dv = new DataView(dt);  return dv;
   }
   void Page_Load(Object sender, EventArgs e) {
      MyDataGrid.DataSource = CreateDataSource();
      MyDataGrid.DataBind();
   }
</script>
<body>
   <form runat=server>
     <ASP:DataGrid id="MyDataGrid" runat="server"  BorderWidth="1"
GridLines="Both"   CellPadding="3"  CellSpacing="0"  />
  </form>
</body>
</html>
```

Value Entero	Valor String
0	Item 0
1	Item 1
2	Item 2

5.4.2.7 Control DataList

Este control muestra datos en una lista repetida, y opcionalmente soporta selección y edición de los ítems. El contenido de la lista de ítems en el control es definido usando *templates*. Como mínimo cada **DataList** debe definir un **ItemTamplate**, como sea, algunos *templates* pueden ser usados para personalizar la apariencia de la lista.

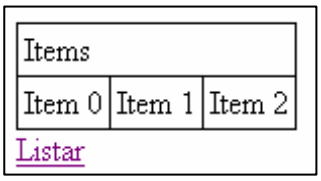
Los templates definen los elementos HTML y controles que pueden ser mostrados para un ítem.

```
<%@ Import Namespace="System.Data" %>
<html>
<script language = "C#" runat="server">
   ICollection CreateDataSource() {
      DataTable dt = new DataTable();
      DataRow dr;
      dt.Columns.Add(new DataColumn("StringValue", typeof(string)));
      for (int i = 0; i < 3; i++) {
         dr = dt.NewRow();
         dr[0] = "Item " + i.ToString();
         dt.Rows.Add(dr);
      }
      DataView dv = new DataView(dt);
      return dv;
   }

   void Page_Load(Object Sender, EventArgs E) {
      if (!IsPostBack) {
         DataList1.DataSource = CreateDataSource();
         DataList1.DataBind();
      }
   }

   void Button1_Click(Object Sender, EventArgs E) {
         DataList1.RepeatDirection = RepeatDirection.Horizontal;
         DataList1.RepeatLayout = RepeatLayout.Table;
         DataList1.BorderWidth = Unit.Pixel(1);
         DataList1.GridLines = GridLines.Both;
   }
</script>
```

```
<body>
 <form runat=server>
 <asp:DataList id="DataList1" runat="server" BorderColor="black"
CellPadding="3"  >
   <HeaderTemplate> Items</HeaderTemplate>
    <ItemTemplate>
        <%# DataBinder.Eval(Container.DataItem, "StringValue") %>
     </ItemTemplate>
  </asp:DataList>
 <asp:LinkButton id=Button1 Text="Listar" OnClick="Button1_Click"
runat="server"/>
   </form>
</body>
</html>
```



5.4.2.8 Control DropDownList

Este control provee una lista de selección simple y desplegable. El siguiente ejemplo muestra su uso.

```
<html>
<head>
   <script language="C#" runat="server">
      void SubmitBtn_Click(Object Sender, EventArgs e) {
         Label1.Text="Escogiste: " + DropDown1.SelectedItem.Text;
      }
   </script>
</head>
<body>
   <form runat=server>
      <asp:DropDownList id=DropDown1 runat="server">
         <asp:ListItem>Item 1</asp:ListItem>
         <asp:ListItem>Item 2</asp:ListItem>
         <asp:ListItem>Item 3</asp:ListItem>
      </asp:DropDownList>
      <asp:button text="Enviar" OnClick="SubmitBtn_Click" runat=server/>
      <asp:Label id=Label1 font-name="Verdana" font-size="10pt"
runat="server">Seleccione un valor de la lista </asp:Label>
   </form>
</body>
</html>
```

5.4.2.9 Control HyperLink

Este control es usado para la navegación del cliente a otra página. El siguiente ejemplo muestra su uso:

```
<html>
<script language="C#" runat=server>
  void Page_Load(Object sender, EventArgs e) {
    HyperLink1.NavigateUrl = "http://www.go-mono.com";
  }
</script>
<body>
   <form runat=server>
      <asp:hyperlink id=HyperLink1 runat="server">
         Vamos al sitio oficial de Mono!!
      </asp:hyperlink>
   </form>
</body>
</html>
```

5.4.2.10 Control Image

El control **Image**, muestra la imagen definida por la propiedad **ImageUrl**. El siguiente ejemplo ilustra su uso.

```
<form runat=server>
  <asp:Image ID="Image1" ImageUrl="/images/mono.gif" AlternateText="Mono"
runat="server" />
</form>
```

5.4.2.11 Control ImageButton

Así como el control **Button**, **ImageButton** es usado para enviar información al servidor. A continuación tenemos un ejemplo

```
<html>
<head>
   <script language="C#" runat="server">
      void ImageButton1_OnClick(object Source, ImageClickEventArgs e) {
        Label1.Text="Presionaste el control  image button";
      }
   </script>
</head>
<body>
```

```
    <form runat=server>
        <asp:ImageButton id=Button1 ImageUrl="/images/mono.jpg"
BorderWidth="2px" onclick="ImageButton1_OnClick" runat="server"/>
        <asp:Label id=Label1 runat=server />
    </form>
</body>
</html>
```

5.4.2.12 Control Label

El control **Label** muestra texto en una sección de la página. Diferente al texto estático, la propiedad **Text** de un **Label** puede ser puesta programáticamente. El siguiente ejemplo muestra su uso.

```
<asp:label id="mensaje1" font-size="16" font-bold="true" forecolor="red"
runat=server>Mensaje 1</asp:label>

 <asp:label id="mensaje2" font-size="20" font-italic="true" forecolor="blue"
runat=server>Mensaje 2</asp:label>

 <asp:label id="mensaje3" font-size="24" font-underline="true" forecolor="green"
runat=server>Mensaje 3</asp:label>
```

5.4.2.13 Control LinkButton

Al igual que el control **Button**, este control es usado para enviar formas Web al servidor. Ejemplo:

```
<html>
<head>
    <script language="C#" runat="server">
      void LinkButton1_Click(Object sender, EventArgs e) {
         Label1.Text="Presionaste el control  link button";
      }
    </script>
</head>
<body>
   <form runat=server>
     <asp:LinkButton Text="Presioname!" Font-Name="Verdana" Font-Size="14pt"
onclick="LinkButton1_Click" runat="server"/>
      <asp:Label id=Label1 runat=server />
    </form>
</body>
</html>
```

5.4.2.14 Control ListBox

Este control provee una lista de simple selección o múltiple selección. Para habilitar la selección múltiple, se ubica la propiedad **SelectionMode** a **Multiple**. El siguiente ejemplo nos muestra su utilización.

```
<html>
<head>
   <script language="C#" runat="server">
      void SubmitBtn_Click(Object Sender, EventArgs e) {
         if (ListBox1.SelectedIndex > -1) {
            Label1.Text="Escogiste: " + ListBox1.SelectedItem.Text;
         }
      }
   </script>
</head>
<body>
   <form runat=server>
      <asp:ListBox id=ListBox1 Width="100px" runat="server">
         <asp:ListItem>Item 1</asp:ListItem>
         <asp:ListItem>Item 2</asp:ListItem>
      </asp:ListBox>
      <asp:button Text="Enviar" OnClick="SubmitBtn_Click" runat="server" />
      <asp:Label id=Label1 font-name="Verdana" font-size="10pt"
runat="server"/>
 </form>
</body>
</html>
```

5.4.2.15 Control Panel

El control **Panel** es un contenedor de otros controles. Es especialmente muy utilizado cuando se quiere generar controles programáticamente, o cuando se necesita ocultar o mostrar un grupo de controles, como en el siguiente ejemplo:

```
<html>
<head>
  <script language="C#" runat="server">
    void Page_Load(Object sender, EventArgs e) {
      Panel1.Visible=false;
       int numlabels = int.Parse(DropDown1.SelectedItem.Value);
       for (int i=1; i<=numlabels; i++) {
         System.Web.UI.WebControls.Label l = new
System.Web.UI.WebControls.Label();
            l.Text = "Label" + i.ToString();
            l.ID = "Label" + i.ToString();
            Panel1.Controls.Add(l);
            Panel1.Controls.Add(new LiteralControl("<br>"));
         }
      }
   </script>

</head>
<body>
   <form runat=server>
      <asp:Panel id="Panel1" runat="server" BackColor="gainsboro"
         Height="200px"  Width="300px">Panel1: Contenido …
      </asp:Panel>
      Generate Labels:
      <asp:DropDownList id=DropDown1 runat="server">
         <asp:ListItem Value="0">0</asp:ListItem>
         <asp:ListItem Value="1">1</asp:ListItem>
      </asp:DropDownList>
    </form>
</body>
</html>
```

5.4.2.16 Control PlaceHolder

El control **PlaceHolder** puede ser usado como un contenedor en un documento para cargar dinámicamente otros controles. Este control no tiene salida basada en HTML y es usado únicamente para marcar un lugar donde otros controles pueden ser añadidos a la colección de controles del **PlaceHolder** durante la ejecución de la página. A continuación un ejemplo:

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Web.UI.HtmlControls" %>
<script language="C#" runat="server">
   void Page_Load(Object sender, EventArgs e) {
      HtmlButton clicker = new HtmlButton();
      clicker.InnerText = "Button 1";
      Bullseye.Controls.Add(clicker);

      clicker = new HtmlButton();
      clicker.InnerText = "Button 2";
      Bullseye.Controls.Add(clicker);
   }
</script>
<html>
<body>
   <asp:PlaceHolder id="Bullseye" runat="server" />
</body>
</html>
```

5.4.2.17 Control RadioButton

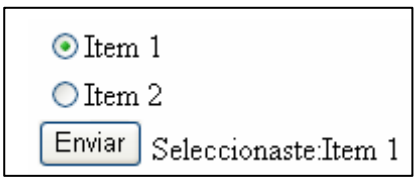
El control **RadioButton** nos permite entremezclar los botones tipo radio en un grupo con otros contenidos en la página. En el ejemplo siguiente, los botones están agrupados lógicamente ya que pertenecen al mismo nombre de grupo.

```
<html>
<head>
   <script language="C#" runat="server">
      void SubmitBtn_Click(Object Sender, EventArgs e) {
         if (Radio1.Checked) {
            Label1.Text = "Seleccioanste: " + Radio1.Text;
         }
         else if (Radio2.Checked) {
            Label1.Text = "Seleccionaste: " + Radio2.Text;
          }
      }
   </script>
</head>
<body>
   <form runat=server>
      <asp:RadioButton id=Radio1 Text="Tipico" Checked="True"
GroupName="RadioGroup1" runat="server" /><br>
      <asp:RadioButton id=Radio2 Text="Compacto" GroupName="RadioGroup1"
runat="server"/><br>
      <asp:button text="Enviar" OnClick="SubmitBtn_Click" runat=server/>
      <asp:Label id=Label1 font-bold="true" runat="server" />
  </form>
</body>
</html>
```

5.4.2.18 Control RadioButtonList

El control **RadioButtonList** provee una lista de selección simple. Como otros controles de lista, este control tiene una colección de ítems cuyos miembros corresponden a cada ítem en la lista.

```
<html>
<head>
   <script language="C#" runat="server">
      void Button1_Click(object Source, EventArgs e) {
         RadioButtonList1.RepeatLayout = RepeatLayout.Table;
         RadioButtonList1.RepeatDirection = RepeatDirection.Vertical;
         if (RadioButtonList1.SelectedIndex > -1)
            Label1.Text = "Seleccionaste:" + RadioButtonList1.SelectedItem.Text;
      }
   </script>
</head>
<body>
   <form runat=server>
      <asp:RadioButtonList id=RadioButtonList1 runat="server">
         <asp:ListItem>Item 1</asp:ListItem>
         <asp:ListItem>Item 2</asp:ListItem>
      </asp:RadioButtonList>
    <asp:Button id=b1 Text="Enviar" onclick="Button1_Click" runat="server"/>
      <asp:Label id=Label1 runat="server"/>
   </form>
</body>
</html>
```



5.4.2.19 Control Repeater

El control **Repeater** muestra un conjunto de datos en una lista de repetición. Similar al control **DataList**, el contenido y el trazado de la lista de

ítems en el control **Repeater** se define usando *templates*. Como mínimo, cada repeater debe definir un **ItemTemplate**.

Diferente al control **DataList**, este control no construye trazados o estilos. Para esto se debe declarar explícitamente todos los trazados HTML y las etiquetas de estilo entre los *templates* del control. Por ejemplo, para crear una lista en una tabla HTML, se debe declarar la etiqueta de la tabla (<*table*>) en el *template* cabecera, las etiquetas de filas y columnas (<*tr*> y <*td*>) en el *template* principal y por último la etiqueta de fin de la tabla (</*table*>) en el *template* de pie.

A continuación tenemos un ejemplo en el que se muestra el uso del control **DataList**:

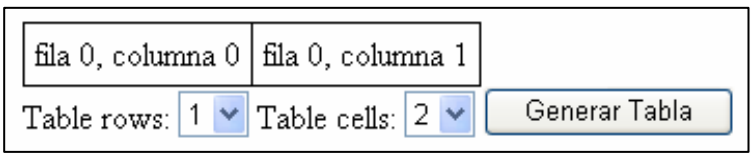
```
<asp:Repeater id=Repeater1 runat="server">
    <HeaderTemplate>
          <table border=1>
            <tr>
              <td><b>Compañía</b></td>
              <td><b>Símbolo</b></td>
            </tr>
     </HeaderTemplate>
     <ItemTemplate>
          <tr>
            <td> <%# DataBinder.Eval(Container.DataItem, "Name") %> </td>
            <td> <%# DataBinder.Eval(Container.DataItem, "Ticker") %> </td>
          </tr>
      </ItemTemplate>
      <FooterTemplate>
          </table>
      </FooterTemplate>
</asp:Repeater>
```

5.4.2.20 Control Table, TableRow y TableCell

El control **Table** crea una tabla programáticamente añadiendo **TableRows** a la colección de filas, y **TableCells** a la colección de columnas de una fila. Se puede añadir contenido a una celda de la tabla programáticamente

añadiendo controles a la colección de controles de la celda. A continuación tenemos un ejemplo:

```
<html>
<head>
   <script language="C#" runat="server">
      void Page_Load(Object sender, EventArgs e) {
         int numrows = int.Parse(DropDown1.SelectedItem.Value);
         int numcells = int.Parse(DropDown2.SelectedItem.Value);
         for (int j=0; j<numrows; j++) {
            TableRow r = new TableRow();
            for (int i=0; i<numcells; i++) {
               TableCell c = new TableCell();
               c.Controls.Add(new LiteralControl("fila " + j.ToString() + ", columna " +
i.ToString()));
               r.Cells.Add(c);
            }
            Table1.Rows.Add(r);
         }
      }
   </script>
</head>
<body>
   <form runat=server>
      <asp:Table id="Table1" CellPadding=5 CellSpacing=0 BorderColor="black"
BorderWidth="1" Gridlines="Both" runat="server"/>
      Table rows:
      <asp:DropDownList id=DropDown1 runat="server">
         <asp:ListItem Value="1">1</asp:ListItem>
         <asp:ListItem Value="2">2</asp:ListItem>
      </asp:DropDownList>
      Table cells:
      <asp:DropDownList id=DropDown2 runat="server">
         <asp:ListItem Value="1">1</asp:ListItem>
         <asp:ListItem Value="2">2</asp:ListItem>
      </asp:DropDownList>
      <asp:button Text="Generar Tabla" runat=server/>
   </form>
</body>
</html>
```



5.4.2.21 Control TextBox

El control **TextBox** permite al usuario ingresar texto. Por defecto, el modo del texto esta en línea simple, pero se puede modificar la característica del **TextBox** ubicando la propiedad **TextMode** a **Password** o **Multiline**.

El ancho de presentación del **TextBox** esta determinado por la propiedad **Columns**. Si **TextMode** está ubicado en **Multiline**, la presentación del **TextBox** se determina por la propiedad **Rows**.

```
<html>
<head>
   <script language="C#" runat="server">
      void SubmitBtn_Click(Object Sender, EventArgs e) {
         Label1.Text = "Text1.Text = " + Text1.Text;
      }
   </script>
</head>
<body>
   <form runat="server">
     <asp:TextBox id="Text1" Text="Copia este texto al label" Width="200px"
runat="server"/>
     <asp:Button OnClick="SubmitBtn_Click" Text="copia text al label"
Runat="server"/>
     <asp:Label id="Label1" Text="Label1" runat="server"/>
   </form>
</body>
</html>
```

5.4.2.22 Control XML

El control **XML** puede ser usado para añadir un documento XML o el resultado de una transformación XML. La propiedad **DocumentSource** especifica el documento XML a usar. Este documento será escrito directamente al flujo de salida a menos que la propiedad **TransformSource** este también especificada. **TransformSource** debe ser un documento valido **XSL Transform** y será usado para transformar el documento XML antes de

que su contenido sea escrito al flujo de salida. A continuación tenemos un ejemplo:

```
<%@ Page Language="C#" %>
<html>
<body>
   <form runat=server>
      <asp:Xml id="xml1" DocumentSource="people.xml"
TransformSource="peopletable.xsl" runat="server" />
   </form>
</body>
</html>
```

Archivo people.xml

```
<People>
   <Person>
      <Name>
         <FirstName>Jhon</FirstName>
         <LastName>Peterson</LastName>
      </Name>
      <Address>
         <Street>Bolivar #450</Street>
         <City>Ibarra</City>
      </Address>
      <Job>
         <Title>Ingeniero Eléctrico</Title>
      </Job>
   </Person>

   <Person>
      <Name>
         <FirstName>Linda</FirstName>
         <LastName>Sue</LastName>
      </Name>
      <Address>
         <Street>Av. Amazonas #4566</Street>
         <City>Quito</City>
      </Address>
      <Job>
         <Title>Ginecologa</Title>
      </Job>
   </Person>
</People>
```

Archivo peopletable.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/People">
     <xsl:apply-templates select="Person" />
  </xsl:template>

  <xsl:template match="Person">
    <table width="18%" border="1">
      <tr>
        <td> <b><xsl:value-of select="Name/FirstName" /> <xsl:value-of
select="Name/LastName" /></b> </td>
      </tr>
      <tr>
        <td><xsl:value-of select="Address/Street" /><br />
          <xsl:value-of select="Address/City" /></td>
      </tr>
      <tr>
        <td>Job Title: <xsl:value-of select="Job/Title" /></td>
      </tr>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

Jhon Peterson
Bolivar #450 Ibarra
Job Title: Ingeniero Eléctrico
Linda Sue
Av. Amazonas #4566 Quito
Job Title: Ginecologa

5.5 USANDO CONTROLES DE SERVIDOR PERSONALIZADOS

ASP.NET cuenta con 45 controles de servidor que pueden ser usados en las páginas Web. Adicionalmente, se puede usar controles desarrollados por vendedores terceros

El siguiente ejemplo muestra un simple control de calendario. Este control **Calendar** es declarado en la página Web usando la etiqueta **<acme:calendar runat=server>** . En la parte superior de la página se tiene la

directiva **<% Register %>** la cual es la responsable de registrar la etiqueta prefija XML "*Acme*" con el namespace "*Acme*" de la implementación del control. El parser de la página ASP.NET utilizará este namespace para cargar la instancia de la clase del control **Calendar** en el tiempo de ejecución.

```
<%@ Register TagPrefix="Acme" Namespace="Acme" Assembly="Acme" %>
<html>
  <body>
    <form action="intro7.aspx" method="post" runat="server">
      <Acme:Calendar id="MiCalendario" runat=server/>
    </form>
  </body>
</html>
```

En el ejemplo anterior, el control **Calendar** ha sido diseñado para un procesamiento de alto nivel en el browser Internet Explorer 5.5 o superior y procesamiento a bajo nivel en todos los otros browsers. Es decir, para Internet Explorer 5.5 o superior genera salida *DHTML*. Esta salida *DHTML* no requiere regresar al servidor cuando se haga una selección en el control. Para todos los otros browsers, el control genera salida estándar *HTML 3.2.* Este código *HTML 3.2* requiere regresar al servidor para tomar las interacciones del usuario en el lado del cliente.

Sin embargo, el código que el desarrollador escribe es igual independiente de que browser se utilice para acceder a la página. El control por si mismo encapsula toda la lógica requerida para tomar los dos escenarios.

5.6 VALIDACIÓN EN CONTROLES DE SERVIDOR

La infraestructura de ASP.NET provee un conjunto de controles de servidor de validación que permiten un uso fácil pero poderoso para validar errores en campos de entrada, y si es necesario, mostrar mensajes al usuario.

Los controles de validación son adheridos a una página ASP.NET como otros controles de servidor. Hay controles para tipos específicos de validación, como puede ser revisar ingresos de valores, lo cual asegura que el usuario no

se salte un campo de ingreso. Se puede añadir más de un control de validación a un control de entrada. Por ejemplo, se puede especificar tanto que una entrada es requerida así como también que debe contener un rango específico de valores.

Los controles de validación trabajan con un limitado conjunto de controles HTML y de servidor Web. Para cada control, una propiedad específica contiene el valor a ser validado. La siguiente tabla lista los controles de entrada que pueden ser validados:

Control	Propiedad de Validación
HtmlInputText	Value
HtmlTextArea	Value
HtmlSelect	Value
HtmlInputFile	Value
TextBox	Text
ListBox	SelectedItem.Value
DropDownList	SelectedItem.Value
RadioButtonList	SelectedItem.Value

5.6.1 Tipos de Controles de Validación

La forma más simple de validación es un campo requerido. Si el usuario ingresa cualquier valor en un campo, este es válido. Si todos los campos en la página son validos, la página es valida.

Igualmente, los controles de validación tienen soporte a clientes de alto nivel y bajo nivel. Para los browsers de alto nivel, realizan la validación en el cliente (usando JavaScript y DHTML) y en el servidor. Los browsers de bajo nivel realizan la validación únicamente en el servidor. El modelo de programación para los dos escenarios es idéntico.

La siguiente tabla lista los controles de validación existentes:

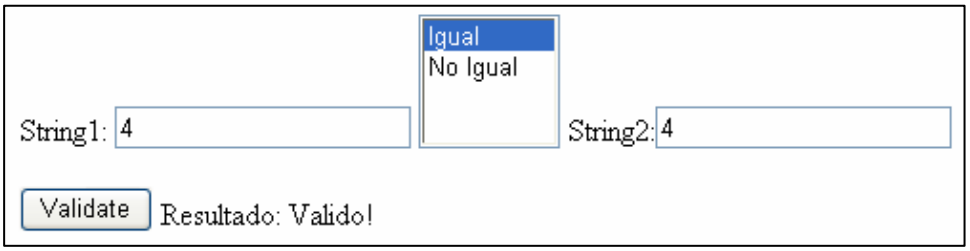
Nombre del control	Descripción
RequiredFieldValidator	Asegura que el usuario no se salte un ingreso
CompareValidator	Compara un entrada del usuario con una valor constante o un valor de una propiedad de otro control usando un operador de comparación (menor que, igual a, mayor que, etc.)
RangeValidator	Revisa que el ingreso de un usuario esté entre límites inferiores y superiores específicos. Se puede chequear rangos entre pares de números, caracteres alfabéticos o fechas. Los límites pueden ser expresados como constantes.
RegularExpressionValidator	Chequea que la entrada corresponda a un modelo definido por una expresión regular. Este tipo de validación permite revisar secuencias de caracteres predecibles, como pueden ser los números de cédula, direcciones de e-mail, números de teléfono, códigos postales, etc.
CustomValidator	Chequea la entrada del usuario usando una validación lógica que codifica usted mismo. Este tipo de validación permite revisar valores derivados en tiempo de ejecución.
ValidationSummary	Muestra los errores de validación en un formulario sumario para todas las validaciones en una página.

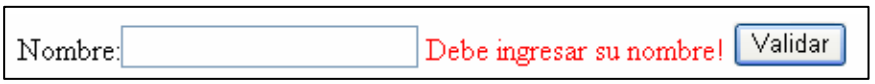
El siguiente ejemplo usa el control **RequiredFieldValidator** en una página para validar el contenido de un **TextBox.**

```
<%@ Page Language="C#" %>
   <script language="C#" runat="server">
      void valida(Object sender, EventArgs e) {       }
  </script><html>
<head>
</head>
<body>
<form action="intro11.aspx" method="post" runat="server">
  Nombre:<asp:textbox id="Nombre" runat="server"/>
  <asp:RequiredFieldValidator ControlToValidate="Nombre" Display="Dynamic"
errormessage="Debe ingresar su nombre!" runat=server/>
  <asp:Button runat=server Text="Validar" ID="Button1" onclick="valida" />
</form></body>
</html>
```

El siguiente ejemplo muestra el uso del control **CompareValidator** en el que se validan dos controles **Text Box**

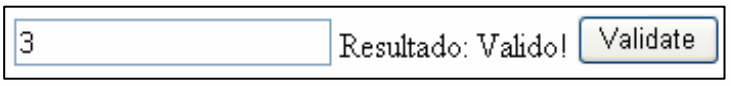
```
<html>
<head>
   <script language="C#" runat="server">
      void validar(Object sender, EventArgs e) {
         if (Page.IsValid) {l.Text = "Resultado: Valido!";}
         else {l.Text = "Resultado: No valido!";}
      }
      void ver(Object sender, EventArgs e) {
         comp1.Operator = (ValidationCompareOperator) lst.SelectedIndex;
         comp1.Validate();
      }
  </script>
</head>
<body>
    <form runat=server>
    String1: <asp:TextBox Selected id="t1" runat="server"></asp:TextBox>
     <asp:ListBox id="lst" OnSelectedIndexChanged="ver" runat="server">
     <asp:ListItem Selected Value="Equal" >Igual</asp:ListItem>
     <asp:ListItem Value="NotEqual" >No Igual</asp:ListItem>
     </asp:ListBox>
    String2:<asp:TextBox id="t2" runat="server"></asp:TextBox><p>
    <asp:Button runat=server Text="Validate" ID="Button1" onclick="validar" />
    <asp:CompareValidator id="comp1" ControlToValidate="t1" ControlToCompare =
"t2" Type="String" runat="server"/>
     <asp:Label ID="l" runat="server"/>
   </form>
</body>
</html>
```





El siguiente ejemplo muestra el uso del control **RangeValidator**

```
<html>
<head>
   <script language="C#" runat="server">
      void B1_Click(Object sender, EventArgs e) {
        rv.Validate();
        if (rv.IsValid) {
          label.Text = "Resultado: Valido!";
          }
        else {
          label.Text = "Resultado: No Valido!";
         }
      }
  </script>
</head>
<body>
  <form runat="server">
     <asp:TextBox Selected id="txt" runat="server"/>
     <asp:Label id="label" runat="server" />
     <asp:Button Text="Validate" ID="B1" onclick="B1_Click" runat="server" />
    <asp:RangeValidator id="rv" Type="Integer" ControlToValidate="txt"
MaximumValue="10" MinimumValue="1" runat="server"/>
</form>
</body>
</html>
```



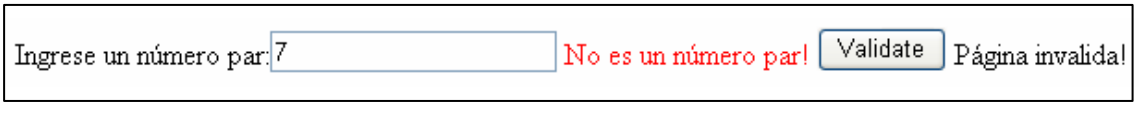
Un ejemplo del uso del control **RegularExpressionValidator es:**

```
<html>
<head>
   <script language="C#" runat=server>
      void ValidateBtn_Click(Object Src, EventArgs E) {
        if (Page.IsValid) {
          label.Text = "La página es Valida!";}
        else {
            label.Text = "La página es Invalida!";
          }
      }
  </script>
</head>
<body>
```

```
<form runat="server">
  <asp:Label ID="label" Text="Ingrese 5 digitos" runat="server"/>
  <asp:TextBox id="TextBox1" runat=server />
  <asp:RegularExpressionValidator id="RegularExpressionValidator1"
runat="server" ControlToValidate="TextBox1" ValidationExpression="^\d{5}$"
          Display="Static"> Debe ingresar solo 5 dígitos numéricos
       </asp:RegularExpressionValidator>
<ASP:Button text="Validate" OnClick="ValidateBtn_Click" runat=server />
 </form>
</body>
</html>
```

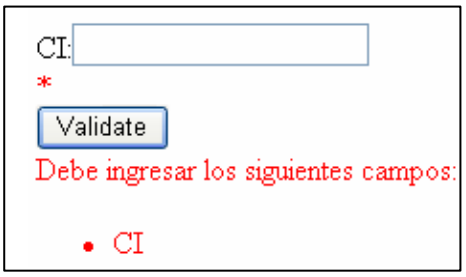
El siguiente ejemplo muestra el uso del control **CustomValidator**

```
<html>
<head>
   <script language="C#" runat=server>
      void ValidateBtn_OnClick(object sender, EventArgs e) {
         if (Page.IsValid) {l.Text = "Página valida!";   }
         else { l.Text = "Página invalida! ";  }
      }
      void ServerValidate (object source, ServerValidateEventArgs value) {
            int num = Int32.Parse(value.Value);
            if (num%2 == 0) {
              value.IsValid = true;
              return;
             }
            else{
                value.IsValid = false;
              return;
             }
      }
  </script>
</head>
<body>
<form runat="server">
   Ingrese un número par:<asp:TextBox id=Text1 runat="server" />
   <asp:CustomValidator id="CustomValidator1" runat="server"
      ControlToValidate="Text1"  OnServerValidate="ServerValidate"
      Display="Static">No es un número par!
   </asp:CustomValidator>
   <asp:Button text="Validate" onclick="ValidateBtn_OnClick" runat="server" />
   <asp:Label id="l" runat="server"/>
</form>
</body>
</html>
```



El siguiente ejemplo muestra el uso del control **ValidationSummary:**

```
<html>
 <body>
  <form runat="server">
    CI:<ASP:TextBox id=TextBox1 runat=server />
    <asp:RequiredFieldValidator id="RequiredFieldValidator2"
        ControlToValidate="TextBox1" ErrorMessage="CI"
        Display="Static"  Width="100%" runat=server>*
    </asp:RequiredFieldValidator>
    <asp:Button id=Button1 text="Validate" runat=server />
    <asp:ValidationSummary ID="valSum" runat="server"
     HeaderText="Debe ingresar los siguientes campos:" />
 </form>
</body>
</html>
```



5.7 CÓDIGO DETRÁS DE LAS FORMAS WEB

ASP.NET soporta dos métodos para realizar páginas dinámicas:

- El primer método es añadir el código físicamente en el archivo *.aspx* original como se ha mostrado en los ejemplos anteriores.
- Otro método alternativo es hacer que la página del código sea más claramente separada del contenido HTML en un archivo separado.

El siguiente ejemplo muestra como usar un archivo separado para el código ASP.NET.

```
<%@ Page Inherits="MyCodeBehind" Src="codigo.cs" %>
<html>
<body>
    <form runat="server">
       <h3> Nombre: <asp:textbox id="Nombre" runat="server"/>
        Categoria:  <asp:dropdownlist id="Categoria" runat=server>
                    <asp:listitem>Cultura</asp:listitem>
                    <asp:listitem>Negocios</asp:listitem>
                    <asp:listitem>Arte</asp:listitem>
                    </asp:dropdownlist>
        </h3>
        <asp:button text="Busqueda" OnClick="b" runat="server"/>
        <asp:Label runat="server" id="Message" />
    </form>
 </body>
</html>
```

```
//archivo codigo.cs
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

public class MyCodeBehind : Page {
   public TextBox      Nombre;
   public DropDownList Categoria;
   public Label Message;

      public void b(Object sender, EventArgs e) {
         if (Page.IsValid) {
          Message.Text = "Hola " + Nombre.Text + ", escogiste " +
Categoria.SelectedItem;
          }
      }
}
```

Además podemos insertar contenido de un archivo especifico en una página ASP.NET, esto se lo hace con la directiva *#Include*. El siguiente ejemplo demuestra como insertar un contenido personalizado:

```
<html>
   <!-- #include virtual="file.inc" -->
<head>
</head>
<body>
   <form runat="server">
          <p>
        <asp:Button id="Button1" runat="server" Text="oK"></asp:Button>
      </p>
   </form>
</body>
</html>
```

Archivo: file.inc

```
<h3>FIN DEL CAPITULO
      </h3>
      <p>
         <strong>:-)</strong>
      </p>
```

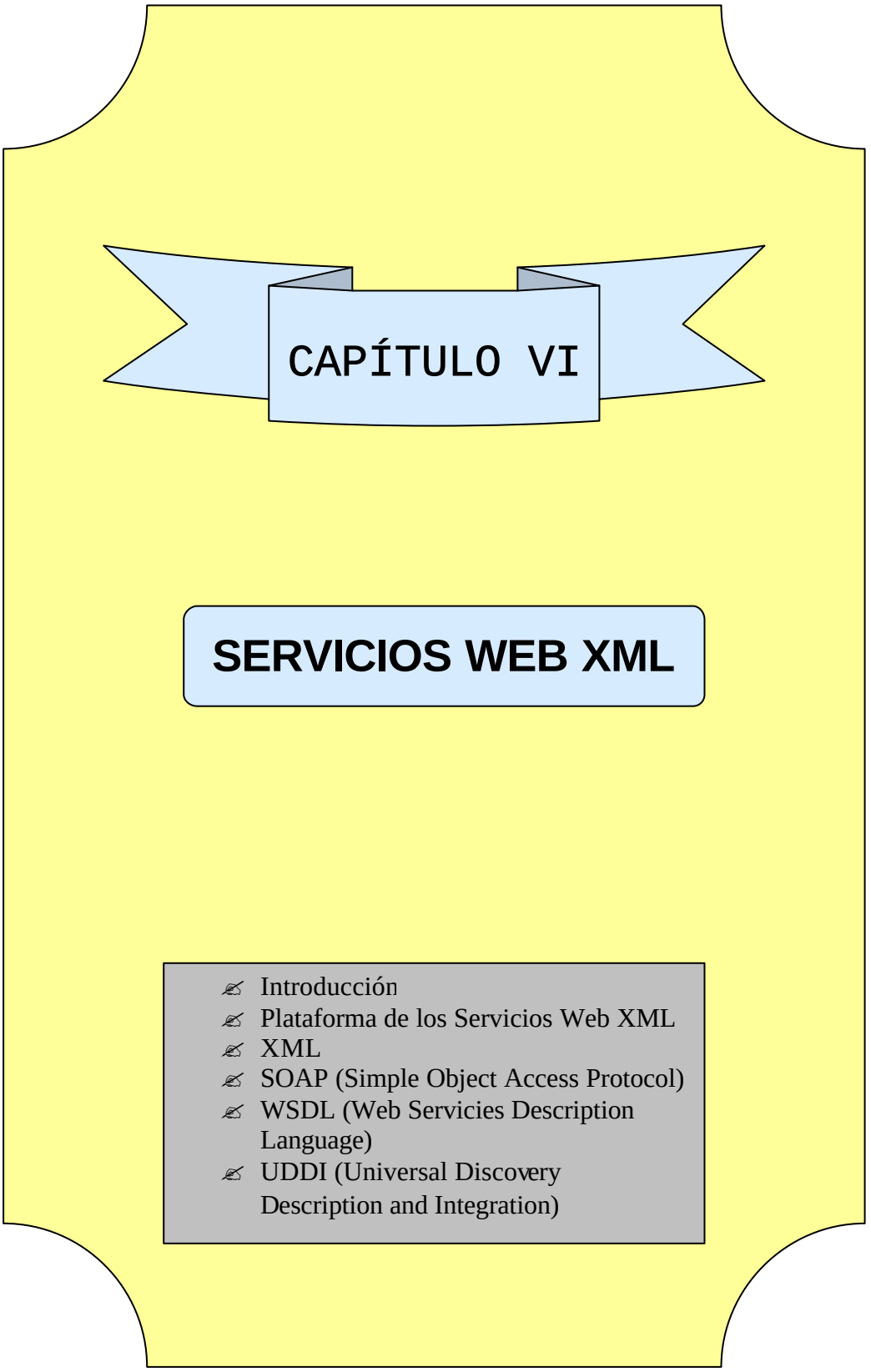
Bibliografía

Libros

- YOUNG, Michael “Aprenda XML Ya”, Editorial Mc Graw Hill, 2000
- FONSECA, Gustavo, “E-Commerce con Linux”, Editorial Prantice Hall, 2001

Refrencias WWW

- http://www.asp.net/Tutorials/quickstart.aspx
- http://www.go-mono.com/asp-net.html
- http://www.gotmono.com/docs/aspnet/introduction.html
- http://www.asp101.com/articles/matt/securesite/default.asp


CAPÍTULO VI
SERVICIOS WEB XML
Introducción
Plataforma de los Servicios Web XML
XML
SOAP (Simple Object Access Protocol)
WSDL (Web Servicies Description Language)
UDDI (Universal Discovery Description and Integration)

6.1 INTRODUCCIÓN

Los servicios Web XML son los bloques de construcción fundamentales en el camino hacia la computación distribuida sobre el Internet. Estándares abiertos, la atención centrada en la comunicación y la colaboración de mucha gente han creado un ambiente donde los servicios Web XML se están convirtiendo en la plataforma para la integración de aplicaciones. Las aplicaciones son construidas usando múltiples servicios Web XML de varios orígenes que trabajan juntos independientemente de donde residan o como estén implementados.

Existen muchas definiciones de un Servicio Web XML, así como hay muchas compañías construyéndolos, pero la mayoría de las definiciones tienen estas cosas en común:

- Los Servicios Web XML exponen su utilidad muy funcional a los usuarios Web a través de un protocolo Web estándar. En muchos casos, el protocolo usado es **SOAP** (*Simple Object Access Protocol*).
- Los Servicios Web proveen una manera de describir sus interfases con sobra de detalles para permitir a un usuario construir una aplicación cliente que los llame. Esta descripción es usualmente usada en un documento XML llamando a un documento con **WSDL** (*Web Services Description Language*).
- Los Servicios Web XML están registrados, así que sus potenciales usuarios pueden hallarlos fácilmente. Esto se logra a través de **UDDI** (*Universal Discovery Description and Integration*).

Una de las ventajas primarias de la arquitectura de los servicios Web XML es que permite que los programas estén escritos en diferentes lenguajes sobre diferentes plataformas para comunicarse con otros en una forma basada en estándares.

Antes ya se había hablado acerca de esto con **CORBA**. La principal diferencia es que **SOAP** es significativamente menos complejo que las anteriores tecnologías, así que los obstáculos para ingresar a una compilación estándar con **SOAP** es significativamente más bajo.

Otra ventaja que los servicios Web XML tienen sobre los esfuerzos previos es que ellos trabajan con protocolos Web Estándar (**XML**, **HTTP**, **TCP/IP**). Muchas compañías cuentan con una infraestructura Web y gente con conocimiento y experiencia en su mantenimiento, así que, el costo para ingresar a los servicios Web XML es mucho menor que las tecnologías previas.

En general, se puede definir a un servicio Web XML como un software de servicio expuesto sobre el Web a través de **SOAP**, descrito con un archivo **WSDL** y registrado en **UDDI**.

Figura 6.1: Componentes de los servicios Web

A los servicios Web XML se los utiliza como orígenes de información que se pueden incorporar fácilmente a las aplicaciones (cotización de existencias,

pronósticos del tiempo, resultados deportivos, etc.). Es fácil imaginar el mundo de aplicaciones que pueden ser construidas para analizar y agregar la información que más nos importa y presentarlo en una variedad de formas. Por ejemplo, se tiene una hoja de *Excel* con información financiera. Si esta información es accesible a través de servicios Web XML, Excel podría actualizarlo continuamente. Alguna de esta información será gratuita y otra podría requerir una suscripción al servicio. La mayoría de esta información se encuentra en la Web, pero los servicios Web XML harán un acceso programático más fácil y más confiable.

Exponer las aplicaciones existentes como servicios Web XML permitirá a los usuarios construir aplicaciones nuevas y poderosas. Por ejemplo, un usuario puede desarrollar una aplicación para compras para obtener automáticamente los precios de muchos proveedores, permitir al usuario seleccionar un proveedor, enviar la orden y entonces rastrear la mercadería hasta que sea recibida. La aplicación del proveedor, adicionalmente a exponer sus servicios en la Web, podría usar los servicios Web XML para revisar el crédito de los clientes, cargar la cuenta de los clientes y enviar la mercadería a través de una compañía de transporte.

En el futuro, algunos de los más interesantes servicios Web XML soportarán aplicaciones que utilizan al Web para hacer todas las cosas que no pueden hacerse hoy. Por ejemplo, uno de los servicios que los servicios Web XML hacen posibles es un calendario. Si su dentista y su mecánico exponen sus calendarios a través de servicios Web XML, se podría crear citas con ellos en línea.

6.2 PLATAFORMA DE LOS SERVICIOS WEB XML

La plataforma básica es **XML** más **HTTP**. **HTTP** es un protocolo que corre prácticamente sobre todo el Internet. **XML** provee un metalenguaje en el cual se puede escribir lenguajes especializados para expresar interacciones complejas entre clientes y servicios o entre componentes de un servicio.

Detrás de la fachada de un servidor Web, un mensaje **XML** se convierte en un requerimiento intermedio y el resultado se convierte de regreso a **XML**.

La Web necesita ser argumentada con algunas otras plataformas de servicios, con lo cual se pueda mantener la simplicidad de la Web, para constituir una plataforma más funcional. La plataforma completa de los servicios Web XML se compone de **XML** más **XML Schema** (Esquemas XML) más **HTTP** más **SOAP** más **WSDL** más **UDDI**.

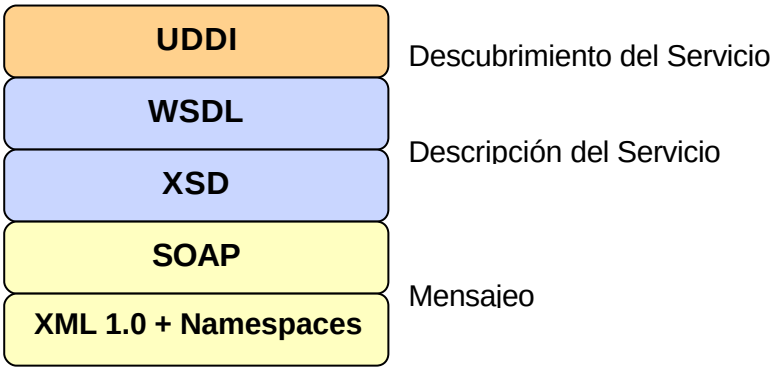


Figura 6.2: Plataforma de los servicios Web

6.3 XML

El lenguaje de marcado extensible (Extensible Markup Language - XML) fue originalmente creado como un lenguaje para definir nuevos formatos de documentos para la Web. XML tiene un formato basado en texto y provee mecanismos para describir estructuras de documentos usando etiquetas de marcado (palabras acompañadas de '<' y '>'). Los desarrolladores de aplicaciones Web pueden notar la similitud entre **HTML** y XML.

Así como el uso de XML ha crecido, ahora es generalmente aceptado que XML no es únicamente útil para describir nuevos formatos de documentos para la Web, sino que también es adecuado para describir datos estructurados. Ejemplos de datos estructurados incluyen información que esta típicamente contenida en archivos de configuración y protocolos de red.

XML puede representar fácilmente tanto datos tabulares (ejemplo: datos relacionales) como datos semi estructurados (como una página Web o un documento de negocios).

XML tiene un número de características que han causado su gran aceptación como un formato de representación de datos. XML es extensible, independiente de la plataforma y soporta internacionalización. El hecho de que su formato sea basado en texto significa que cuando las necesidades aumentan, se puede usar cualquier herramienta estándar de edición de textos para modificar el código.

La extensibilidad se manifiesta de varias formas. Primeramente, diferente a **HTML**, no tiene un vocabulario fijo. El programador puede definir vocabularios específicos para aplicaciones particulares usando XML. Segundo, las aplicaciones que procesan o consumen formatos XML son más resistentes a cambios en la estructura de código XML en comparación con aplicaciones que usan otros formatos

XML no está atado a un lenguaje de programación, sistema operativo o software específico. De hecho, es muy simple producir o consumir XML usando una gran variedad de lenguajes. La independencia de la plataforma hace a XML muy útil si se trata de interoperar entre diferentes plataformas de programación y sistemas operativos.

6.3.1 XML y Namespaces

Incluir namespaces es el proceso de usar uno o más elementos o atributos del namespace en un documento XML. Los nombres tanto de elementos como de atributos son realmente compuestos por dos partes: un nombre del namespace y un nombre local. En un documento XML usamos un prefijo namespace para calificar los nombres locales de los elementos y atributos. Un prefijo es una abreviación para el identificador del namespace (*URI- Universal Resource Identifier*), el cual es típicamente muy largo. El prefijo

es primeramente mapeado al identificador del namespace a través de una declaración del namespace. La sintaxis para la declaración de un namespace es:

```
xmlns:<prefijo>='<identificador del namespace>'
```

Supongamos que un desarrollador desea usar el namespace ***XSLT 1.0***. Se necesitará proveer una declaración del namespace que mapee un prefijo arbitrario al identificador oficial del namespace ***XSLT 1.0*** (*http://www.w3.org/1999/XSL/Transform*). Entonces, cada elemento o atributo que el desarrollador desea usar desde ***XSLT 1.0*** simplemente necesita usar el prefijo escrito. El siguiente ejemplo nos muestra esto:

```
<x:transform version='1.0'
  xmlns:x='http://www.w3.org/1999/XSL/Transform'>
  <x:template match='/'>
    <hello_world/>
  </x:template>
</x:transform>
```

XML1.0[38] junto a los Namespaces proveen la sintaxis para la representación de datos. Usando **XML 1.0**, los desarrolladores pueden representar fácilmente datos desde sus sistemas internos en un formato portable. Esto simplifica las cosas ya que si se trata de procesar mensajes, el parser de XML lo hace fácilmente.

Por ejemplo, la información de una calculadora de hipotecas puede ser representada usando el siguiente documento **XML1.0**:

```
<m:Hipoteca  xmlns:m="http://example.org/mortgage">
  <monto>100000</monto>
  <años>30</años>
  <interes>8.0</interes>
  <ImpuestoAnual>1000</ImpuestoAnual>
  <garantiaAnual>300</garantiaAnual>
</m:Hipoteca>
```

[38] XML 1.0 : Información en: http://www.w3.org/TR/REC-xml.html

Note que el documento contiene un elemento “Hipoteca” que representa la operación, y varios elementos hijos que contienen las entradas requeridas: “monto”, “años”, “interés”, “ImpuestoAnual”, ”garantíaAnual”. El mensaje de respuesta puede ser también representado en un documento XML, como muestra el siguiente ejemplo:

```
<m:RespuestaHipoteca  xmlns:m="http://example.org/mortgage">
  <PagosHipotecarios>
    <PIMensual>733.76</PIMensual>
    <ImpuestoMensual>83.33</ImpuestoMensual>
    <GarantialMensual>25</GarantiaMensual>
    <TotalMensual>842.09</TotalMensual>
  </PagosHipotecarios>
</m:RespuestaHipoteca>
```

Usando **XML 1.0** para representar la interfaz hace mucho más fácil para otros programas que consuman la operación. **XML 1.0** es fácil programar a través de una variedad de servicios que conforman el modelo abstracto de **XML Information Set** conocido también como **Infoset**. Estos servicios incluyen **XPath** para consultar documentos, **XSLT** para la transformación de documentos, y **DOM**, **XmlReader** – **XmlWriter**, **XPathNavigator** y otros API’s para trabajar programaticamente con documentos.

El siguiente ejemplo en C# muestra como extraer la entrada de datos usando un API XML en .NET.

```
void processInputStream(Stream s)
{
  double monto, años, interest, annualTax, annualIns;
  XmlReader r = new XmlTextReader(s);
  r.ReadStartElement("Hipoteca","http://example.org/mortgage");
  Monto = XmlConvert.ToDouble(r.ReadElementString("monto"));
  Años = XmlConvert.ToDouble(r.ReadElementString("años"));
  •••
  r.ReadEndElement();
  •••
}
```

6.3.2 XML Schema

Un lenguaje XML Schema es usado para describir la estructura y contenido de un documento XML. Durante el intercambio de un documento, un esquema XML describe el contrato entre el productor y el consumidor de XML empezando por describir que constituye un mensaje XML válido entre las dos partes.

Existen un gran número de lenguajes esquema para XML, pero el único que actualmente aplica con reglas es el ***W3C XML Schema Definition Language***[39] que típicamente se lo abrevia como **XSD**.

XSD es único entre los lenguajes esquema XML porque es el primero en ayudar a expandir el rol de un esquema XML fuera de su tradicional rol de describir el contrato entre dos entidades intercambiando documentos.

A continuación tenemos un ejemplo, donde un esquema es usado para especificar un documento cuyos elementos son discos compactos, los cuales cada uno contiene un precio, título y un elemento artista.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="items">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="compact-disc" minOccurs="0"
maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="compact-disc">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="precio" type="xs:decimal" />
      <xs:element name="artista" type="xs:string" />
      <xs:element name="titulo" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

[39] W3C XML Schema Definition Language : http://www.w3.org/TR/xmlschema-1/

6.3.3 Modelo de árbol basado en API's

Un modelo de árbol API expone a un documento XML como un árbol de nodos, el cual típicamente es cargado en memoria todo de una vez. El más popular modelo de árbol API para XML es el ***W3C Document Object Model (DOM)***[40]. El **DOM** permite la lectura programática, manipulación y modificación de un documento XML.

A continuación tenemos un ejemplo de la clase **XmlDocument** de .NET para obtener el nombre del artista y el título del primer disco compacto de los ítems. Utiliza como lenguaje a C#.

```
using System;
using System.Xml;

public class Test{
  public static void Main(string[] args){
    XmlDocument doc = new XmlDocument();
    doc.Load("test.xml");
    XmlElement firstCD = (XmlElement) doc.DocumentElement.FirstChild;
XmlElement artist  =
(XmlElement) firstCD.GetElementsByTagName("artist")[0];
    XmlElement title   =
(XmlElement) firstCD.GetElementsByTagName("title")[0]
    Console.WriteLine("Artist={0}, Title={1}", artist.InnerText, title.InnerText);
  }
}
```

6.4 SOAP (Simple Object Access Protocol)

SOAP originalmente permaneció como "Simple Object Access Protocol". Hace algunos años se entendía a SOAP como un protocolo que hacía que **DCOM** y **Corba** trabajen sobre el Internet. Los autores originales admiten que se centraron en el acceso a objetos. Actualmente se enfocó la especificación y se movió desde los objetos hacia la estructura de intercambio de mensajes de XML.

El cambio de enfoque creo un problema con la "O" en el acrónimo de

[40] W3C Document Object Model (DOM): información en: http://www.w3.org/DOM/

SOAP. El Grupo de trabajo de SOAP 1.2 mantuvo el nombre original pero cambiaron la definición, la cual ya no menciona a los objetos:

SOAP es un protocolo creado para el intercambio de información estructurada en un ambiente descentralizado y distribuido. SOAP usa la tecnología XML para definir una estructura de intercambio de mensajes, la cual provee un constructor de mensajes que pueden ser intercambiados sobre una variedad de protocolos. La estructura ha sido designada para ser independiente de cualquier modelo de programación en particular y otras semánticas de implementación específicas.

Esta definición realmente llega al corazón de los que actualmente es SOAP. SOAP define una manera de mover mensajes XML desde un punto A a un punto B (*figura 6.3*). Lo hace con una estructura de intercambio de mensajes basada en XML. Esta estructura es extensible, usable a través de una variedad de protocolos de red e independiente de los modelos de programación.

Figura 6.3: Intercambio simple de mensajes SOAP

Cuando hablamos de extensibilidad, podemos ver que en el acrónimo de SOAP tenemos la 'S' de Simple. En lo referente al Web, la simplicidad gana sobre la eficiencia o la técnica pura, y cuando la interoperabilidad está en juego, entonces la simplicidad es muy requerida.

SOAP puede ser usado sobre cualquier protocolo de transporte como puede ser: **TCP**, **HTTP**, **SMTP**, etc. Para mantener la interoperabilidad como

sea, el ligamiento de protocolos estándar necesita definir las reglas para cada ambiente. La especificación SOAP provee una estructura flexible para definir arbitrariamente ligamientos de protocolos y provee un ligamiento explicito para **HTTP** ya que es muy utilizado.

SOAP esta permitido para cualquier modelo de programación y no está ligado a **RPC** (*Remote Procedure Calling*). Mucha gente iguala SOAP a hacer llamadas **RPC** sobre objetos distribuidos, cuando de hecho el modelo fundamental de SOAP es más semejante a sistemas tradicionales de envío de mensajes como **MSMQ** (*Microsoft Message Queuing*)[41]. SOAP define un modelo para procesamiento individual, mensajes en una vía. Se puede combinar múltiples mensajes en un solo intercambio de mensajes. En la figura 6.2 vimos un simple mensaje en una vía donde el emisor no recibe una respuesta. El receptor podría enviar una respuesta de regreso al emisor (*figura 6.4*).

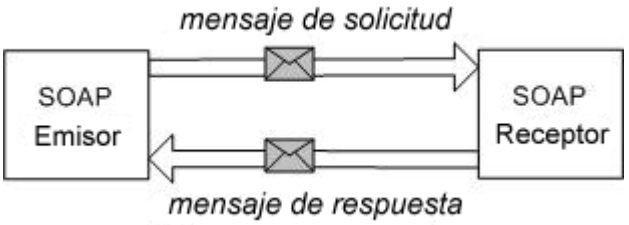


Figura 6.4: Intercambio de mensajes con solicitud - respuesta

Los desarrolladores pueden confundir solicitud / respuesta con **RPC** cuando realmente actúan de manera diferente. **RPC** usa requerimiento / respuesta, pero requerimiento / respuesta no necesariamente usa **RPC**. **RPC** es un modelo de programación que permite a los desarrolladores trabajar con llamadas a métodos.

[41] MSMQ: información en:
http://www.microsoft.com/windows2000/technologies/communications/msmq/default.asp

6.4.1 Estructura de Messaging (Mensajeo)

El corazón de la especificación SOAP es la estructura de messaging. Esta estructura define un conjunto de elementos XML para empaquetar arbitrariamente los mensajes XML para transportarlos entre sistemas.

La estructura consiste en los siguientes elementos XML: **Envelope**, **Header**, **Body** y **Fault**, los cuales vienen del namespace de **SOAP 1.1.** y que se lo puede encontrar en *http://schemas.xmlsoap.org/soap/envelope/* .

La siguiente plantilla de mensaje ilustra la estructura de un **Envelope** SOAP:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header> <!-- optional -->
  <!-- header blocks go here... -->
 </soap:Header>
 <soap:Body>
  <!-- payload or Fault element goes here... -->
 </soap:Body>
</soap:Envelope>
```

El elemento **Envelope** es siempre el elemento raíz de un mensaje SOAP. Para las aplicaciones lo hace muy fácil para identificar los mensajes SOAP simplemente mirando el nombre del elemento raíz.

El elemento **Envelope** contiene un elemento opcional **Header** seguido de elemento mandatario **Body**. Este elemento es un contenedor genérico en el que puede contener cualquier número de elementos desde cualquier namespace. A la larga aquí es donde se encuentran los datos que se está tratando de enviar.

Por ejemplo, el siguiente mensaje SOAP representa un requerimiento para transferir fondos entre cuentas bancarias:

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <x:TransferFunds xmlns:x="urn:examples-org:banking">
   <from>22-342439</from>
   <to>98-283843</to>
   <amount>100.00</amount>
  </x:TransferFunds>
 </soap:Body>
</soap:Envelope>
```

Si el receptor soporta requerimiento / respuesta y si esta permitido para procesar el mensaje, podría enviar otro mensaje SOAP de regreso al emisor inicial. En este caso, la información de respuesta podría también ser contenida en el elemento **Body** como se ilustra en el siguiente ejemplo:

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <x:TransferFundsResponse
   xmlns:x="urn:examples-org:banking">
   <balances>
    <account>
     <id>22-342439</id>
     <balance>33.45</balance>
    </account>
    <account>
     <id>98-283843</id>
     <balance>932.73</balance>
    </account>
   </balances>
  </x:TransferFundsResponse>
 </soap:Body>
</soap:Envelope>
```

La estructura de messaging también define un elemento llamado **Fault** para representar errores en el elemento **Body** cuando las cosas van mal. Esto es esencial porque sin una representación estándar de errores, cada aplicación podría inventar su propia representación y hacer imposible de distinguir entre sucesos y fallas. En el siguiente ejemplo un mensaje SOAP contiene un elemento **Fault** que indica el mensaje “Fondos insuficientes” si hay errores ocurridos mientras se procesa el requerimiento:

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <soap:Fault>
   <faultcode>soap:Server</faultcode>
   <faultstring>Fondos Insuficientes</faultstring>
   <detail>
    <x:TransferError xmlns:x="urn:examples-org:banking">
     <sourceAccount>22-342439</sourceAccount>
     <transferAmount>100.00</transferAmount>
     <currentBalance>89.23</currentBalance>
    </x:TransferError>
   </detail>
  </x:TransferFunds>
 </soap:Body>
</soap:Envelope>
```

La mayoría de los protocolos existentes hacen distinción entre información de control (cabeceras) y el mensaje en si mismo. SOAP también lo hace ya que los elementos **Header** y **Body** proveen la misma distinción con XML. El gran beneficio del **Envelope** es que puede ser usado con cualquier protocolo de comunicaciones.

Las cabeceras siempre han jugado un papel muy importante en los protocolos de aplicación como **HTTP**, **SMTP**, etc., porque permiten que las aplicaciones negocien la conducta de los comandos soportados. Con la especificación SOAP, las cabeceras también juegan un rol muy importante. Cuando las cabeceras SOAP se estandaricen, será muy fácil para los desarrolladores definir protocolos de aplicación, sin la necesidad de reinventarlos cada vez.

6.4.2 Modelo de Procesamiento

SOAP define un modelo de procesamiento donde hay reglas para procesar los mensajes SOAP como es el caso de como viajan desde el emisor SOAP al receptor SOAP. El modelo de procesamiento permite arquitecturas muy interesantes como el que se presenta en la *Figura 6.5*, la cual contiene múltiples nodos intermedios:

Figura 6.5: Mensajeo SOAP sofisticado

Un intermediario se sitúa entre el emisor inicial y el receptor final e intercepta los mensajes SOAP. Un intermediario actúa tanto como emisor y receptor al mismo tiempo. Los nodos intermediarios hacen posible diseñar algunas arquitecturas de red que pueden ser influenciadas por el contenido del mensaje.

Mientras se procesa un mensaje, un nodo SOAP asume uno o más roles que influencian como las cabeceras SOAP son procesadas. Los roles son nombres únicos (en forma de URI's) así que pueden ser identificados durante el procesamiento. Cuando un nodo SOAP recibe un mensaje para procesamiento, primeramente debe determinar que roles asumirá. Este puede inspeccionar el mensaje SOAP para ayudar a hacer esta determinación.

Una vez que determina los roles en los cuales actuará, el nodo SOAP debe entonces procesar las cabeceras mandatorias (marcadas con **mustUnderstand="1"**) destinadas a uno de estos roles. El nodo SOAP puede también escoger procesar cualquier cabecera opcional (marcadas con **mustUnderstand="0"**) destinado a uno de estos roles.

SOAP 1.1. únicamente define un único rol llamado ***http://schemas.xmlsoap.org/soap/actor/next*** (se le llamará **next**). Cada nodo SOAP es requerido para asumir el rol **next**. Por tanto, cuando un mensaje SOAP arribe a cualquier nodo SOAP, el nodo debe procesar todas las cabeceras mandatarias destinadas al rol **next**, y puede escoger las cabeceras opcionales también destinadas al rol **next**. Para **SOAP 1.2** se definen más

roles y las aplicaciones están habilitadas a definir roles personalizados también. Estos roles se definen a continuación:

Nombre del Rol SOAP	Descripción
http://www.w3.org/2002/06/soap-envelope/role/next	Cada SOAP intermediario y el último receptor SOAP debe actuar en este rol y puede asumir adicionalmente otros roles SOAP.
http://www.w3.org/2002/06/soap-envelope/role/none	Los nodos SOAP no deben actuar en este rol.
http://www.w3.org/2002/06/soap-envelope/role/ultimateReceiver	Para establecerse a si mismo como un receptor SOAP final, un nodo SOAP puede actuar en este rol. Los intermediarios SOAP no deben actuar en este rol

6.5 WSDL (Web Services Description Language)

La arquitectura de los servicios Web XML define un mecanismo estándar para hacer que los recursos sean accesibles vía mensajeo XML sobre protocolos estándar como son **TCP**, **HTTP** o **SMTP**. El término Servicio Web, o simplemente servicio, típicamente se refiere a una pieza de código implementando la interfaz XML a un recurso, al cual podría ser difícil de acceder de otra manera (*figura 6.5*)

Figura 6.6: Recursos y servicios

Esta arquitectura hace posible para cualquier consumidor con soporte de XML que se integre a las aplicaciones de servicios Web. Los consumidores deben determinar a priori la interfaz XML junto con otros detalles del mensaje. **XML Schema** puede llenar parcialmente esta necesidad porque permite a los desarrolladores describir la estructura de los mensajes XML. Sin embargo, **XML Schema** de forma solitaria no puede describir los detalles adicionales dentro de la comunicación con un servicio Web. Una definición esquema

simplemente nos dice que mensajes XML pueden ser usados pero no como se relacionan con otros.

Un intercambio de mensajes representa una operación. Los consumidores se preocupan mucho de las operaciones, puesto que estos son el punto focal de interacción con el servicio (*figura 6.6*).

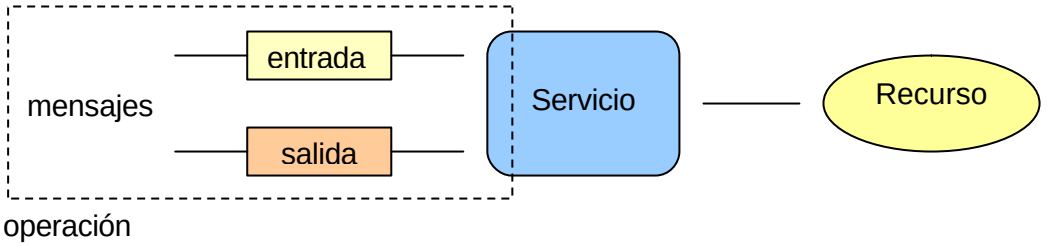


Figura 6.7: Mensajes y operaciones

Para los desarrolladores es común agrupar en interfaces a las operaciones relacionadas. Es importante trabajar con servicios Web en ambientes orientados a objetos ya que las interfaces XML pueden mapear programáticamente las clases abstractas en el lenguaje que se ha escogido.

Los consumidores también deben conocer que protocolo de comunicación se usa para el envío de mensajes al servicio, en compañía de los mecanismos específicos que envuelven al protocolo dado como son comandos, cabeceras o códigos de error.

Un servicio puede soportar múltiples enlaces para una interfaz dada, pero cada enlace debe ser accesible a una dirección única identificada por un URI, también conocido como el punto final del servicio Web (*figura 6.7*).

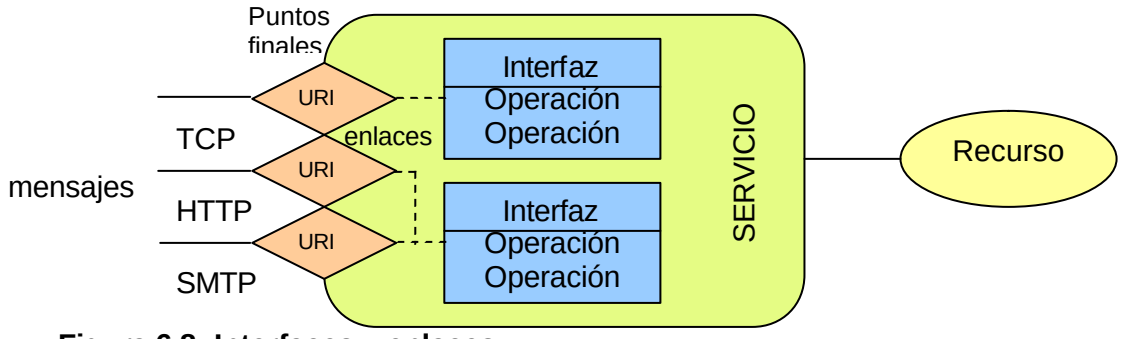


Figura 6.8: Interfaces y enlaces

El WSDL provee una gramática XML para describir todos los detalles, definiendo los enlaces para cada interfaz y combinación de protocolos. Juega un importante papel en la arquitectura de los servicios Web ya que describe el contrato completo para comunicación de la aplicación

Hoy por hoy, los desarrolladores pueden utilizar las definiciones WSDL para generar código que conozca precisamente como interactuar con el servicio Web al que lo describe. Este tipo de generación de código oculta los detalles tediosos envueltos en el envío y recepción de mensajes SOAP sobre diferentes protocolos y hace muy provechosos a los servicios Web.

6.5.1 Elementos de WSDL

Una definición WSDL es un documento XML con un elemento de definición raíz desde el namespace: **http://schemas.xmlsoap.org/wsdl/** . Las definiciones pueden contener algunos otros elementos incluyendo **types**, **message**, **portType**, **binding** y **service** los cuales vienen del namespace **http://schemas.xmlsoap.org/wsdl/**. El siguiente ejemplo ilustra la estructura básica de una definición WSDL:

```
<!– WSDL definition structure -->
<definitions
    name="MathService"
targetNamespace="http://example.org/math/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
>
   <!-- abstract definitions -->
    <types> ...
    <message> ...
   <portType> ...

   <!-- concrete definitions -->
   <binding> ...
   <service> ...
</definition>
```

Note que se debe especificar el namespace destino para nuestra definición WSDL, así como se lo haría para una definición **XML Schema**. Cualquier cosa que se nombre en la definición WSDL (como un **message**,

portType, **binding**, etc.) automáticamente forma parte del namespace destino de la definición WSDL definida por el atributo **targetNamespace**. Por tanto, cuando se referencia algo por el nombre en el archivo WSDL, se debe recordar usar un nombre calificado.

Los primeros tres elementos (**types**, **message**, y **portType**) son todas definiciones abstractas de la interfaz del servicio Web. Estos elementos constituyen la interfaz programática que se añade en el código. Los dos últimos elementos (**binding** y **service**) describen los detalles concretos de cómo la interfaz abstracta mapea los mensajes en la línea. Estos detalles son tomados por la infraestructura en sí, no por nuestro código de aplicación. La siguiente tabla muestra las definiciones para cada uno de estos elementos centrales WSDL.

Nombre del elemento	Descripción
Types	Un contenedor para definiciones de tipo abstractas definidas usando *XML Schema*.
message	Una definición de un mensaje abstracto que puede consistir en múltiples partes, cada una puede ser de diferente tipo.
portType	Un conjunto abstracto de operaciones soportadas por uno o más puntos finales (comúnmente conocido como una interfaz); las operaciones son definidas usando un intercambio de mensajes.
binding	Un protocolo concreto y una especificación de formato de datos para un *portType* particular.
service	Una colección de puntos finales relacionados, donde un punto final es definido como una combinación de un *binding* y una dirección (URI)

6.5.1.1 Types

El elemento **types** de WSDL es un contenedor para las definiciones **type** de **XML Schema**. Las definiciones **type** que se ubican aquí son referenciadas desde definiciones **message** de alto nivel para definir los detalles estructurales del mensaje.

El elemento **types** contiene cero o más elementos esquema del namespace **http://www.w3.org/2001/XMLSchema**. La estructura básica del

elemento **types** (con los namespaces omitidos) es como sigue ("*" significa cero o más):

```
<definitions .... >
   <types>
      <xsd:schema .... />*
   </types>
</definitions>
```

El siguiente fragmento WSDL contiene una definición de **XML Schema** que define elementos para el **type "**MathInput*"* y elementos para el **type** "MathOutput*"*.

```
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:y="http://example.org/math/"
  xmlns:ns="http://example.org/math/types/"
  targetNamespace="http://example.org/math/"
>
  <types>
    <xs:schema
      targetNamespace="http://example.org/math/types/"
      xmlns="http://example.org/math/types/"
    >
     <xs:complexType name="MathInput">
       <xs:sequence>
      <xs:element name="x" type="xs:double"/>
      <xs:element name="y" type="xs:double"/>
       </xs:sequence>
     </xs:complexType>
     <xs:complexType name="MathOutput">
       <xs:sequence>
         <xs:element name="result" type="xs:double"/>
       </xs:sequence>
     </xs:complexType>
    <xs:element name="Add" type="MathInput"/>
     <xs:element name=”AddResponse" type="MathOutput"/>
     <xs:element name="Multiply" type="MathInput"/>
     <xs:element name="MultiplyResponse" type="MathOutput"/>
    </xs:schema>
  </types>
   ...
</definitions>
```

6.5.1.2 Messages

El elemento **message** de WSDL define una mensaje abstracto que puede servir como la entrada o la salida de una operación. Los **messages**

consisten en una o más elementos **part**, donde cada parte es asociada con uno u otro elemento (cuando se usa el estilo documento) o un tipo (cuando se usa el estilo **RPC**). La estructura básica de una definición **message** es como sigue ("*" significa cero o más y "?" significa opcional):

```
<definitions .... >
  <message name="nmtoken"> *
    <part name="nmtoken" element="qname"? type="qname"?/> *
  </message>
</definitions>
```

Los mensajes y las partes deben ser nombrados haciendo posible referirse a ellos desde cualquier parte de la definición WSDL. Si se está definiendo un servicio de estilo **RPC**, las partes del mensaje representan los parámetros de los métodos. En este caso, el nombre de la parte se convierte en el nombre de un elemento en el mensaje concreto y esta estructura está determinada por el atributo **type** suministrado. Si se está definiendo un servicio de estilo documento, las partes simplemente se refieren a los elementos XML que están ubicados entre el cuerpo (referenciado por el atributo **element**). El siguiente ejemplo contiene algunas definiciones de mensajes que se refieren a elementos por nombre:

```
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:y="http://example.org/math/"
  xmlns:ns="http://example.org/math/types/"
  targetNamespace="http://example.org/math/"
>
   ...
   <message name="AddMessage">
     <part name="parameter" element="ns:Add"/>
   </message>
   <message name="AddResponseMessage">
     <part name="parameter" element="ns:AddResponse"/>
   </message>
   <message name="SubtractMessage">
     <part name="parameter" element="ns:Subtract"/>
   </message>
   <message name="SubtractResponseMessage">
     <part name="parameter" element="ns:SubtractResponse"/>
   </message>
   ...
</definitions>
```

6.5.1.3 Interfaces (portTypes)

El elemento **portType** de WSDL define un grupo de operaciones, también conocido como una interfaz en la mayoría de entornos. Desafortunadamente, el término **portType** es confuso, por lo cual es mejor usar el término *interfaz*. Un elemento **portType** contiene cero o más elementos **operation**. La estructura básica de un **portType** es como sigue ("*" significa cero o más):

```
<definitions .... >
   <portType name="nmtoken">
      <operation name="nmtoken" .... /> *
   </portType>
</definitions>
```

Cada elemento **operation** contiene una combinación de elementos **input** y **output**. El uso del elemento **input** seguido de **output** define una operación requerimiento – respuesta. El uso de **output** seguido de **input** define una operación solicitud – respuesta. Solamente el elemento **input** define una operación en un solo sentido (una vía). Solamente el elemento **output** define una operación de notificación.

El siguiente ejemplo define un **portType** llamado "MathInterface" que consiste en dos operaciones matemáticas: "Suma" y "Resta".

```
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:y="http://example.org/math/"
  xmlns:ns="http://example.org/math/types/"
  targetNamespace="http://example.org/math/"
>
   ...
   <portType name="MathInterface">
      <operation name="Suma">
         <input message="y:AddMessage"/>
         <output message="y:AddResponseMessage"/>
      </operation>
      <operation name="Resta">
         <input message="y:SubtractMessage"/>
         <output message="y:SubtractResponseMessage"/>
      </operation>
   </portType>
    ...
</definitions>
```

6.5.1.4 Bindings

El elemento **binding** describe los detalles concretos de usar un **portType** particular con un protocolo dado. El elemento **binding** contiene algunos elementos de extensibilidad como es el elemento **operation** de WSDL para cada operación en el **portType** que esta describiendo. La estructura básica de un elemento **binding** es como sigue (* significa cero o más y ? significa opcional)

```
<wsdl:definitions .... >
   <wsdl:binding name="nmtoken" type="qname"> *
      <-- extensibility element providing binding details --> *
      <wsdl:operation name="nmtoken"> *
        <-- extensibility element for operation details --> *
        <wsdl:input name="nmtoken"? > ?
           <-- extensibility element for body details -->
        </wsdl:input>
        <wsdl:output name="nmtoken"? > ?
           <-- extensibility element for body details -->
        </wsdl:output>
        <wsdl:fault name="nmtoken"> *
           <-- extensibility element for body details -->
        </wsdl:fault>
      </wsdl:operation>
   </wsdl:binding>
</wsdl:definitions>
```

6.5.1.5 Services

Este elemento define una colección de **ports** (puertos) y **endpoints** (puntos finales) que exponen un enlace particular. Su estructura básica es:

```
<definitions .... >
   <service .... > *
      <port name="nmtoken" binding="qname"> *
        <-- extensibility element defines address details -->
      </port>
   </service>
</definitions>
```

Se debe dar a cada puerto un nombre y asignarlo a un **binding**. Entonces, entre el elemento **port** se usa un elemento **extensibility** para definir los detalles específicos de la dirección al **binding**. Por ejemplo, el siguiente

ejemplo define un servicio llamado “MathService” que expone el **binding** “MathSoapHttpBinding” a la URL *http:///localhost/math/math.asmx*

```
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:y="http://example.org/math/"
  xmlns:ns="http://example.org/math/types/"
  targetNamespace="http://example.org/math/"
>
   ...
   <service name="MathService">
      <port name="MathEndpoint" binding="y:MathSoapHttpBinding">
        <soap:address
          location="http://localhost/math/math.asmx"/>
      </port>
   </service>
</definitions>
```

6.6 UDDI (*Universal Discovery Description and Integration*)

UDDI es las páginas amarillas de los servicios Web. Como las tradicionales páginas amarillas, se puede buscar cualquier compañía que ofrezca los servicios que necesitamos, leer acerca del servicio ofrecido y contactar a alguien para más información. Por supuesto que también se puede ofrecer un servicio Web sin registrarlo en UDDI, pero si realmente se desea llegar a un mercado significativo es necesario UDDI para que los clientes puedan encontrarnos.

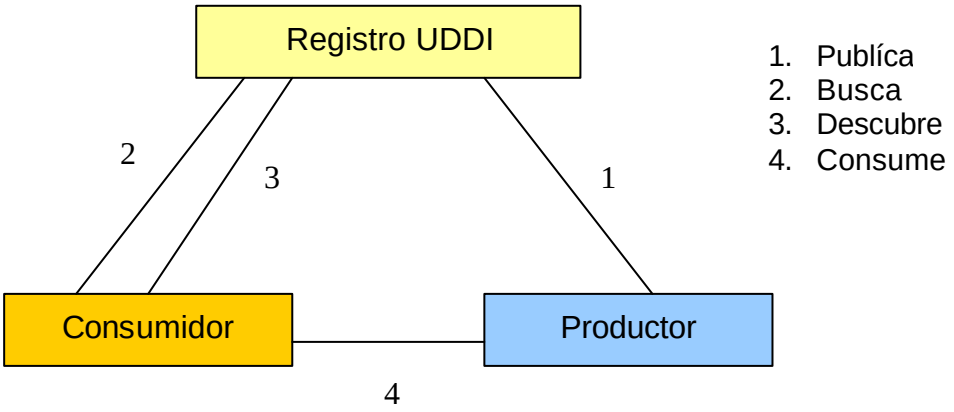


Figura 6.9: Funcionamiento de UDDI

Un registro de un directorio UDDI es un archivo XML que describe un negocio y los servicios que ofrece. Hay tres partes de un registro en el directorio UDDI: las “páginas blancas” que describen a la compañía que ofrece el servicio: nombre, dirección, contactos, etc. Las “páginas amarillas” incluyen categorías industriales basadas en taxonomías estándar como es el Sistema de Clasificación de la Industria Norteamericana (*North American Industry Classification System*) y la Clasificación Industrial Estándar (*Standard Industrial Classification*). Las “páginas verdes” describen la interfaz al servicio en suficiente detalle para alguien que escriba una aplicación que use el servicio Web.

La forma en que los servicios Web son definidos es a través de un documento UDDI llamado **Type Model** o **tModel**. En muchos casos, **tModel** contiene un archivo WSDL que describe una interfaz SOAP para un servicio Web XML, pero el **tModel** es flexible permitiendo describir casi cualquier tipo de servicio.

El directorio UDDI también incluye algunas formas de búsqueda de los servicios que necesitamos construir para nuestras aplicaciones. Por ejemplo, se puede buscar proveedores de un servicio en una locación geográfica específica o de negocios o de un tipo específico. El directorio UDDI entonces suministra información, contactos, enlaces y datos técnicos para permitir evaluar cuales servicios cubren nuestros requerimientos.

La categorización es la característica más importante de UDDI. El propósito de UDDI tanto en tiempo de diseño como en tiempo de ejecución es la habilidad de atribuir metadatos a los servicios registrados en UDDI, y correr consultas basadas en esos metadatos.

UDDI provee metadatos a través de algunos significados: algunas entidades centrales en UDDI pueden ser adornadas con lo que podría llamarse bolsas de propiedades que no es más que colecciones de pares (nombre -

valor) que describen la entidad dada. Cada propiedad en la bolsa viene de un sistema de clasificación conocido. Por ejemplo, un servicio puede ser adornado con una bolsa la cual aclara lo que el servicio es: localizado en Ecuador, permitido para ser usado las 24 horas del día en los 7 días de la semana y es un servicio bancario.

La propiedad que aclara que este servicio esta situado en Ecuador viene de un sistema de clasificación geográfico, la propiedad acerca del nivel de servicio viene de un sistema de clasificación de calidad con un conjunto diferente de valores, y así sucesivamente. Adornar a las entidades UDDI con estas bolsas de propiedades provee entidades con contextos y metadatos críticos que pueden ser usados para descubrirlos y consumirlos.

Otras características hacen de la arquitectura de UDDI capaz de ubicar un rango de escenarios. Por ejemplo, las consultas pueden hacer un exacto emparejamiento de todas las propiedades en una bolsa o pueden emparejar solo una propiedad en la bolsa. La capacidad de consulta in el API UDDI provee una gran cantidad de flexibilidad en términos de escritura enfocada y consultas precisas.

Lo que hace a UDDI particularmente poderoso es que no hay límites en el número de diferentes esquemas de clasificación que pueden ser creados. Se puede crear un ilimitado número de esquemas de clasificación e ilimitado número de valores para ese esquema. El contexto para ese esquema es completamente determinado por la persona quien los crea.

Por ejemplo, puede haber tres diferentes esquemas de clasificación de calidad del servicio que tienen diferentes valores y significados. O puede haber múltiples esquemas geográficos: uno para donde está el servicio físicamente, otro para las áreas que el servicio cubre, etc.

En UDDI, estas bolsas o colecciones son conocidas como elementos **categoryBag** y las propiedades con conocidas como elementos **keyedReference**.

Los metadatos se atribuyen a entidades UDDI en elementos **keyedReference**. Este elemento contiene tres atributos: **keyValue**, **keyName** y **tModelKey**. El elemento **keyValue** contiene la propiedad principal de búsqueda, mientras que el elemento **keyName** es para un entendimiento de los programadores. El elemento **tModelKey** es usado para conocer de que esquema de categorización viene esa propiedad.

El siguiente ejemplo muestra a una compañía que se ha categorizado como un proveedor localizado en Ibarra.

```
(1)<?xml version="1.0" encoding="utf-8"?>
(2)<businessEntity businessKey="Ã‚Â…" xmlns="urn:uddi-org:api_v2">
(3) <name>Company</name>
(4) <categoryBag>
(5)  <keyedReference
(6)     tModelKey="uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2"
(7)     keyName="NAICS: Software Publisher"
(8)     keyValue="51121" />
(9)   <keyedReference
(10)     tModelKey="uuid:4e49a8d6-d5a2-4fc2-93a0-0411d8d19e88"
(11)     keyName="California"
(12)     keyValue="US-CA" />
(13) </categoryBag>
(14)</businessEntity>
```

El elemento **categoryBag** en (4) puede contener un ilimitado número de elementos **keyedReference**. Hay dos elementos **keyedReference** (5) y (9) cada uno con diferente elemento **tModelKey**. Este **tModelKey** es el identificador de un **tModel** único que existe en el registro UDDI. Los elementos **tModel** son constructores usados en UDDI para representar conceptos u otros constructores. En términos de base de datos, se puede comparar a un **tModel**

como una clave foránea que se refiere a otra tabla. En este caso, el elemento **tModelKey** identifica a un único esquema de clasificación.

Para saber que sistema de clasificación esta representado por un **tModelKey**, se usa una llamada API UDDI para buscar el **tModel** basado en este **tModelKey**. Comparándolo en términos de base de datos, esto equivalente a rastrear la tabla de donde la clave foránea se deriva. En UDDI, la llamada API apropiada para ser usada es **get_tModelDetail** en el cuerpo de un mensaje SOAP como sigue:

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <get_tModelDetail generic="2.0" xmlns="urn:uddi-org:api_v2">
      <tModelKey>uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2</tModelKey>
      <tModelKey>uuid:4e49a8d6-d5a2-4fc2-93a0-0411d8d19e88</tModelKey>
    </get_tModelDetail>
  </Body>
</Envelope>
```

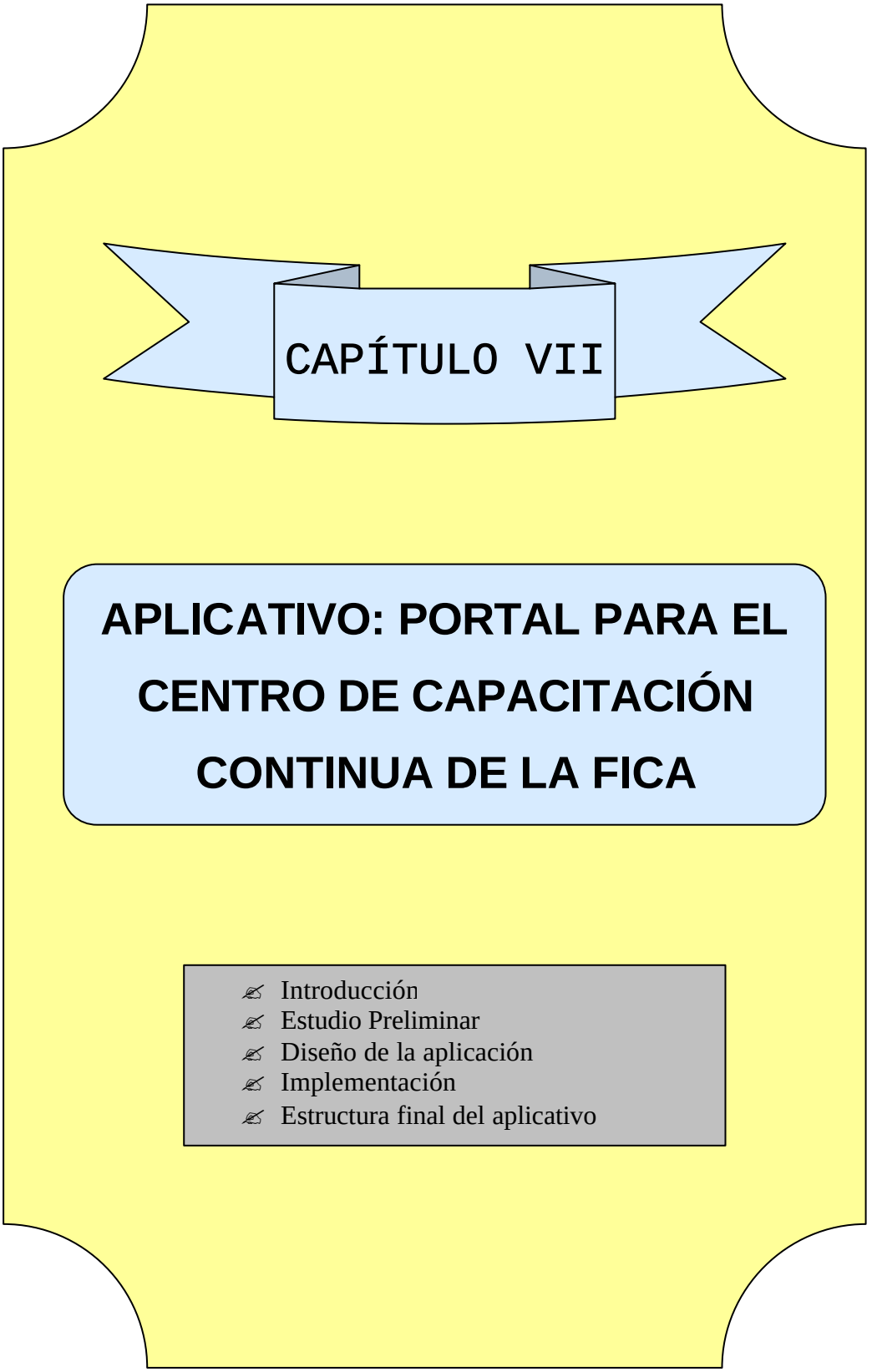
Cada identificador es pasado como un valor del elemento **tModelKey**. El resultado de esta llamada API retorna el **tModelDetail** para estos dos esquemas de clasificación. Como resultado a tal llamada, se descubre que en (6) el **tModelKey** se refiere al Sistema de Clasificación Industrial de Norteamérica (*North American Industry Classification System - NAICs*) mientras que en (8), el **tModelKey** se refiere a la Taxonomía Geográfica ISO 3166.

Retornando al ejemplo, podemos ver que el valor “51121” del atributo **keyValue** en (8), es una pieza de metadato que está asociado con la entidad del esquema de clasificación de NAIC. El tributo **keyName** en (7) provee un contexto de lo que el atributo **keyValue** en realidad significa. El atributo **keyName** es enteramente usado para un entendimiento humano contextual, y no tiene relevancia en términos de búsqueda o consulta. Los únicos valores importantes en una búsqueda o consulta son los atributos **keyValue** y **tModelKey**.

Bibliografía

Referencias WWW

- http://www.alistapart.com/articles/webservices/
- http://msdn.microsoft.com/webservices/understanding/
- http://www.sys-con.com/webservicesedge2003east/86.cfm
- http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html
- http://www.sw.nec.co.jp/middle/WebOTX-e/function/webservice.html
- http://www.w3.org/TR/xml11/
- http://www.w3.org/TR/wsdl


CAPÍTULO VII
APLICATIVO: PORTAL PARA EL CENTRO DE CAPACITACIÓN CONTINUA DE LA FICA
Introducción
Estudio Preliminar
Diseño de la aplicación
Implementación
Estructura final del aplicativo

7.1 INTRODUCCIÓN

El Portal para el Centro de Capacitación Continua, nace de la necesidad de los estudiantes de la Facultad de Ingeniería en Ciencias Aplicadas y público en general de informarse de una manera más eficiente acerca de los cursos, seminarios y eventos que se ponen a disposición en la Facultad de Ingeniería en Ciencias Aplicadas de la Universidad Técnica del Norte.

Es muy importante tener un centro automatizado donde se concentre toda la información concerniente a los diferentes cursos, seminarios y eventos que se expongan en la FICA, todo esto con la finalidad de lograr una organización y un mejor manejo de los procesos que muchas veces se transforman en papeleo que se ubica por todas partes, carteleras con boletines que muestran la información de cursos o eventos por realizarse pero que muchas veces no cubren toda la información que debe mostrarse y que es necesaria para tomar la decisión de inscribirse en un curso o seminario.

El objetivo de este capítulo no es mostrar manuales técnicos y de usuario detallados, sino el de informar los pasos necesarios para diseñar e implantar una aplicación bajo la plataforma de desarrollo .NET utilizando el software Mono en un sistema operativo Linux.

7.2 ESTUDIO PRELIMINAR

Como primer paso, fue necesario buscar un aplicativo que pueda tomar todas las potencialidades de la plataforma .NET y concentrarlas en un solo sistema de manera que se pueda mostrar las características más potentes de esta nueva plataforma de desarrollo.

Con la idea inicial de las autoridades de la FICA de crear el Centro de Capacitación Continua de la FICA, surgió la necesidad de crear un sistema que permita concentrar toda la información de este nuevo centro y por supuesto que sea de gran ayuda para los estudiantes de la facultad para que

puedan interactuar con un sistema que les permita conocer los cursos, seminarios y eventos de la FICA.

ASP.NET forma parte de la gran plataforma de desarrollo .NET, con lo cual es posible crear aplicaciones basadas en Web. Actualmente todas las computadoras con cualquier sistema operativo integran browsers que permiten mostrar páginas Web. Es por esta razón que se decidió hacer un portal basado en Web ya que sería muy fácil de acceder por parte de los usuarios. Además, dependiendo de las necesidades empresariales, un sitio Web puede ser ubicado en un servidor Web que puede funcionar tanto a nivel local en una Intranet, como también puede funcionar con salida a Internet lo que lo haría accesible desde cualquier parte del mundo.

Como plataforma para la realización de este aplicativo se pensó muy claramente desde el principio en el sistema operativo Linux, debido a las grandes potencialidades que ofrece como servidor Web y como servidor de base de datos concentrando su mayor ventaja en la seguridad frente a otras plataformas.

La idea inicial fue que el sistema muestre información acerca de:

- Cursos
- Seminarios
- Eventos

La información que debe mostrar debe ser lo mas detallada posible, de manera que los estudiantes interesados en un curso o seminario puedan informarse totalmente y puedan tomar la decisión de pre-inscribirse en el instante en que leen la información que muestra el sistema.

Esta información debe incluir:

- Nombre y descripción del curso o seminario
- Objetivos del curso o seminario

- Precio y las horas que durará la capacitación
- La fechas de iniciación y finalización del curso
- Los respectivos temas, subtemas y horarios del curso o seminario
- Información acerca del instructor, lo cual incluye su hoja de vida.
- Los requisitos para inscribirse en el curso o seminario

Es necesario permitir al usuario que se pre-inscriba en una curso o seminario y además asignarle una cuenta de usuario con el fin de que en un futuro pueda revisar sus notas, escribir comentarios o sugerencias, etc.

Por último, es muy importante el módulo de administración que permita el ingreso, actualización y eliminación de cursos, seminarios, eventos, instructores, etc., así como tener reportes de alumnos inscritos, reportes históricos de cursos, etc.

7.3 DISEÑO DE LA APLICACIÓN

Luego de hacer un estudio inicial de las necesidades de la Facultad de Ingeniería en Ciencias Aplicadas de la Universidad Técnica del Norte, es necesario empezar a diseñar la aplicación, para lo cual se debe analizar dos aspectos básicos que posteriormente serán el armazón de la aplicación durante la implementación de la misma. Estos dos aspectos son: diseño de la interfaz de usuario y diseño de la base de datos.

7.3.1 Diseño de la Interfaz de Usuario

Para diseñar una interfaz de usuario para un sistema basado en Web, es indispensable pensar en el sin número de enlaces que puede tener. Se debe pensar primeramente en las características principales que tendrá el sistema y luego a partir de esto se debe ir construyendo tantos enlaces a otras páginas Web como sea necesario.

Por tanto, para empezar a diseñar la Interfaz de Usuario basada en Web se debe considerar las características principales del Marco principal:

- Colores y formas en base a los colores distintivos de la empresa o institución para la cual se diseña el sitio Web.
- Logotipos de la institución a mostrar.
- El titulo principal del sitio Web.
- Banners y animaciones que se mostraran.
- Determinar como será el cuerpo principal de las páginas Web donde se mostrará la información relevante.
- Menú principal que se mostrará y los enlaces que debe contener.

Una vez que se deciden todos estos parámetros, se debe empezar a diseñar la interfaz de usuario del sitio Web para lo cual es necesario utilizar herramientas especiales para diseño y edición de imágenes.

Como software de diseño de la interfaz de usuario, se utilizó las herramientas de Macromedia: **Flash MX**, **Dreamweaver 4.0** y **Fireworks 4.0**.

Flash MX es una herramienta que permite realizar animaciones, banners y menús. Con esta herramienta se diseño los banners publicitarios que se sitúan en la parte izquierda de la pantalla.

Dreamweaver 4.0 es una herramienta que permite mantener un sitio Web en perfecto orden y además permite enviar los archivos a un servidor vía ftp o vía red de área local.

Fireworks 4.0 es una herramienta que permite la edición de imágenes. Con esta herramienta se diseñó todo el contorno de la página y además se editaron las imágenes que contiene el sitio Web.

A continuación se muestra el marco principal del sistema:

Figura 7.1: Diseño de la Interfaz de usuario

7.3.2 Diseño de la Base de Datos

Una vez que se tiene diseñado la interfaz de usuario, es necesario realizar el diseño de las tablas que constituirán nuestra Base de Datos.

Para esto primeramente se piensa en las entidades y las relaciones que formarán parte de nuestro sistema. Es muy importante la lluvia de ideas que debe surgir al pensar en las entidades y relaciones del sistema ya que forma la base de lo que a futuro será el diseño físico de la base de datos.

Una herramienta fundamental para nuestro propósito es la utilización de **UML** (Unified Modeling Language) Lenguaje de Modelado Unificado que nos sirve para visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de software.

UML proporciona una forma estándar de escribir los planos de un sistema, cubriendo tanto las cosas conceptuales, tales como los procesos del negocio y funciones del sistema, como las cosas concretas tales como las

clases escritas en un lenguaje de programación específico, esquema de base de datos y componentes de software reutilizables.

Es importante modelar por las siguientes razones:

- El modelado es una parte central de todas las actividades que conducen a la producción de un buen software.
- Construimos modelos para comunicar la estructura deseada y el comportamiento de nuestro sistema.
- Construimos modelos para visualizar y controlar la arquitectura de nuestro sistema.
- Construimos modelos para comprender de mejor manera el sistema que estamos construyendo, muchas veces descubriendo oportunidades para la simplificación y reutilización.

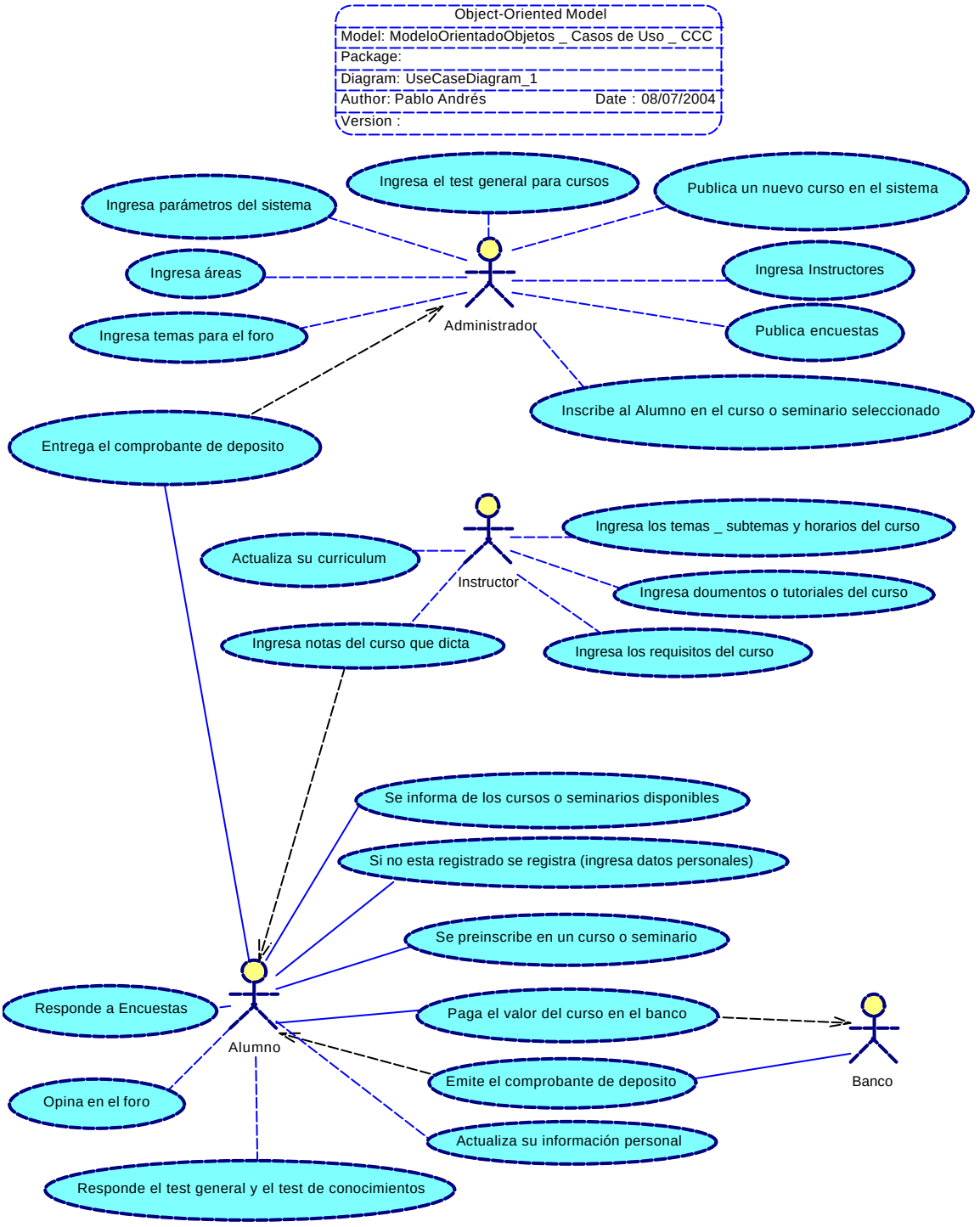
Es decir conseguimos cuatro objetivos básicos:

- Los modelos ayudan a visualizar como es o como queremos que sea un sistema.
- Los modelos nos permiten especificar la estructura o el comportamiento de un sistema.
- Los modelos nos proporcionan plantillas que nos guían en la construcción de un sistema.
- Los modelos documentan las decisiones que hemos tomado.

Para empezar a utilizar este lenguaje de modelado, es importante conocer todos los bloques de construcción que posee UML como son Elementos, Relaciones y Diagramas. Estos elementos no se los revisará en detalle, por tanto es preciso entrar al desarrollo de los modelos en cuestión.

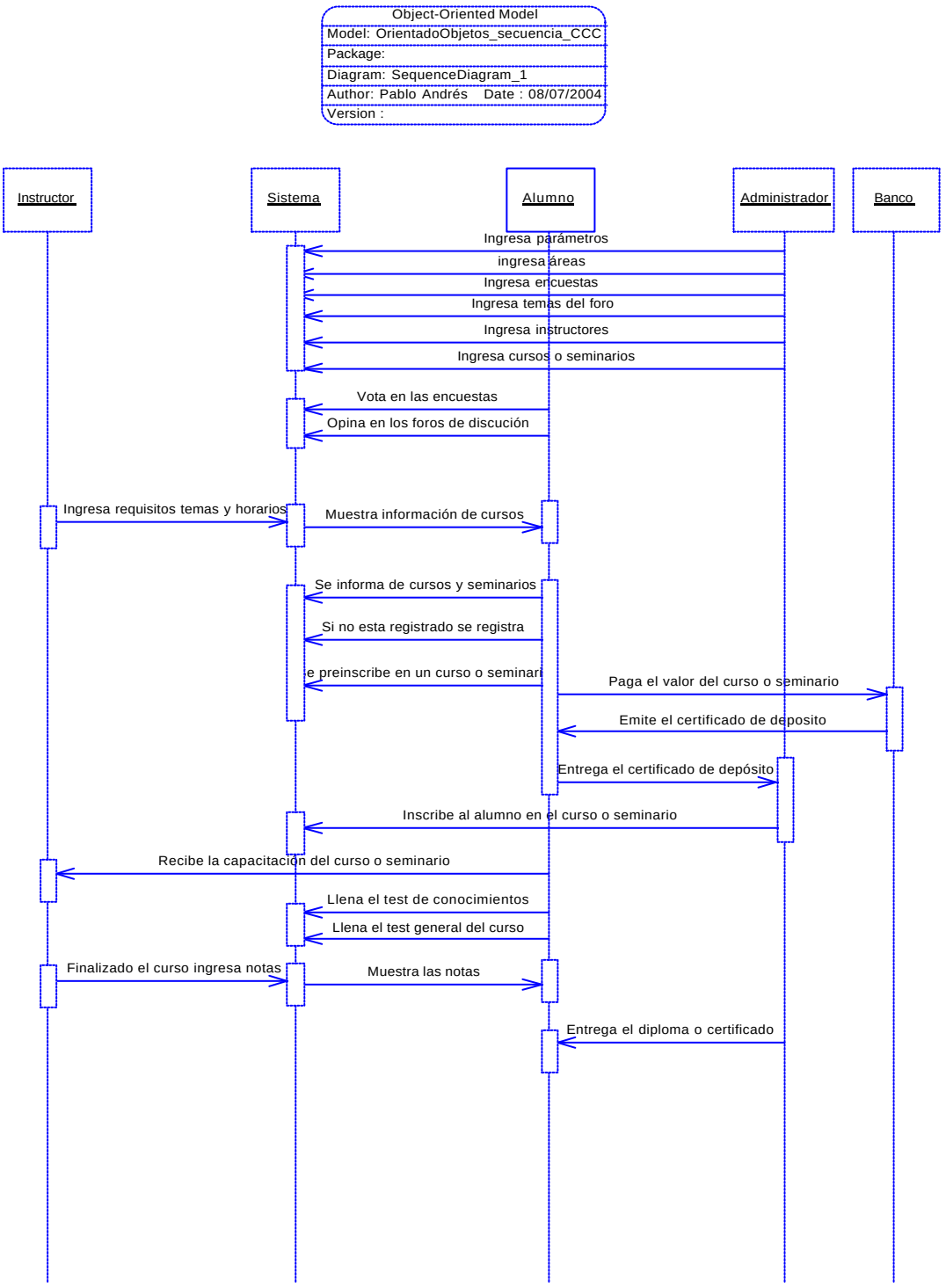
Empezaremos modelando el Diagrama de Casos de Uso, el cual muestra un conjunto de casos de usos, sus actores y sus relaciones. Cubre la vista de diseño estática del sistema.

Diagrama de Casos de Uso – Portal Centro de Capacitación Continua FICA



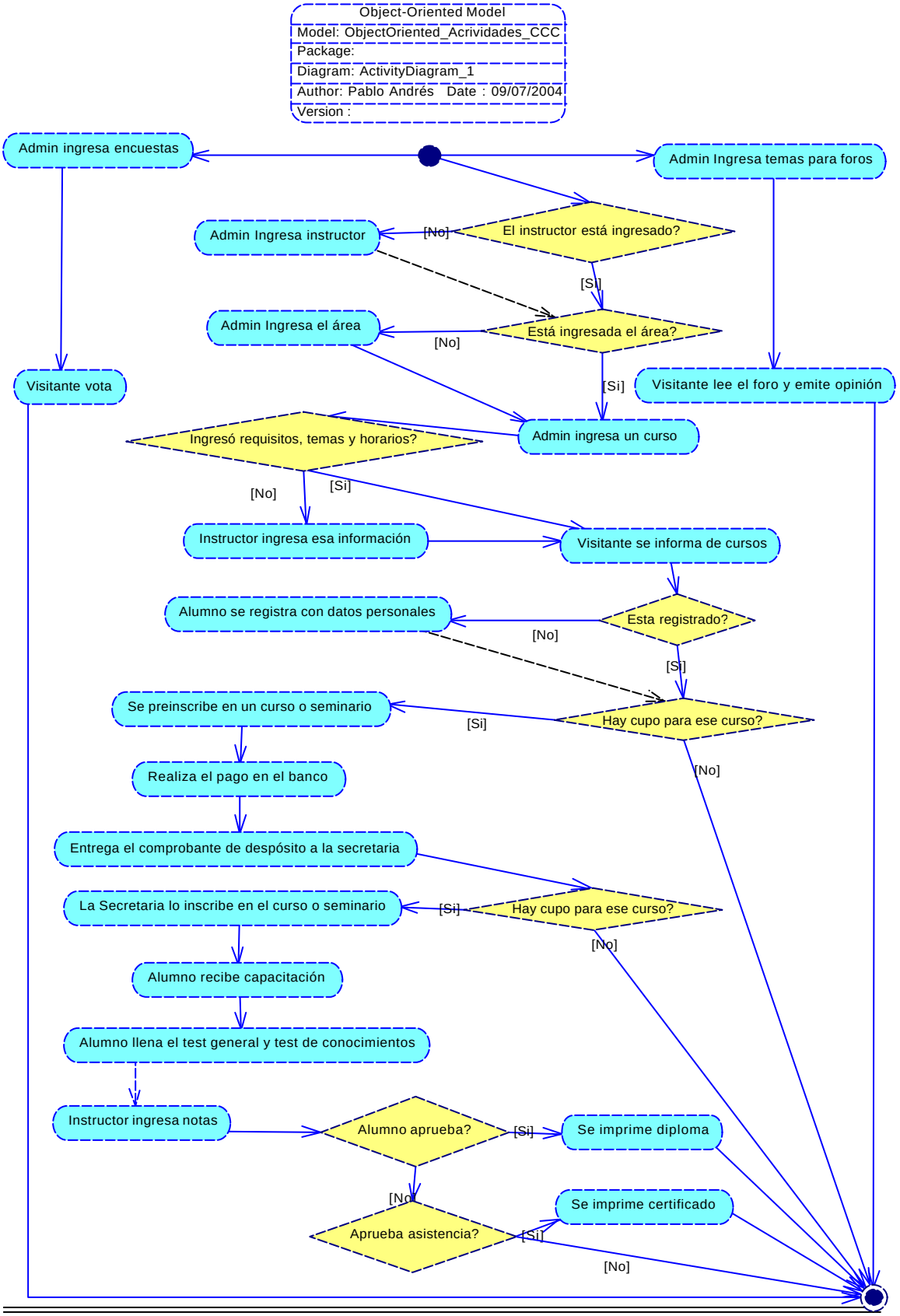
A continuación presentamos el Diagrama de Secuencia, el cual resalta la ordenación temporal de los mensajes. Consta de un conjunto de objetos y sus relaciones y cubre la vista dinámica del sistema.

Diagrama de Secuencia – Portal Centro de Capacitación Continua FICA



A continuación tenemos el Diagrama de Actividades, el cual muestra el flujo de actividades dentro del sistema. Cubre la vista dinámica del sistema.

Diagrama de Actividades – Portal Centro de Capacitación Continua FICA



Una vez que se ha establecido los casos de uso, las secuencias y el flujo de actividades que tendrá el sistema, podemos construir el Diagrama de Clases, donde se muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Este diagrama cubre la vista de diseño estática del sistema.

Diagrama de Clases – Portal Centro de Capacitación Continua FICA

Luego de realizar estos diagramas, podemos concluir con el diseño Físico de la base de datos:

Diseño Físico Base de Datos – Portal Centro de Capacitación Continua FICA

7.4 IMPLEMENTACIÓN

7.4.1 Plataforma

El Portal para el Centro de Capacitación Continua ha sido desarrollado en el Sistema Operativo **Linux Red Hat 8.0**. Este sistema operativo ha demostrado a través de los años todas sus potencialidades especialmente en términos de seguridad y rendimiento, por esta razón es muy utilizado como servidor para aplicaciones muy variadas.

Como software de desarrollo se utilizó a **Mono 0.28**[42] desarrollado por la empresa Ximian, el cual es una implementación de la tecnología **Microsoft .Net** para plataformas de desarrollo libre.

Como parte de la distribución de Mono 0.28 se utilizó: **mcs 0.28** (compilador de C# en Mono), **mono 0.28** (contiene las librerías de clases y el runtime de Mono) y **xsp 0.5**(es un servidor Web simple para ejecutar aplicaciones ASP.NET)**.**

Como Base de Datos, se utilizó a MySQL, siendo esta una base de datos muy liviana lo cual ayuda a una aplicación no tan grande como la que estamos realizando como investigación.

7.4.2 Requerimientos para el computador Servidor

El computador que será servidor Web y de Base de datos debe tener los siguientes requerimientos de hardware:

- Computador Pentium III de 800 Mhz como mínimo. Recomendado Pentium IV de 1.2 Mhz.
- Espacio en Disco Duro de 3 Gb como mínimo. Recomendado espacio en Disco Duro de 8 Gb

[42] Actualmente existe una nueva versión de Mono (Mono 1.0) disponible en http://www.go-mono.com/download.html

- Memoria RAM de 128 Mb como mínimo. Recomendado 192 para modo gráfico.
- Tarjeta de red para la interconexión local

Como requerimientos de software para el servidor tenemos:

- Sistema operativo Linux Red Hat 8.0
- Mono 0.28 (incluye mcs-0.28, mono -0.28 y xsp-0.5).
- Base de Datos MySQL
- Php, si se desea instalar un administrador basado en Web para la base de datos MySQL.

7.4.3 Requerimientos para los computadores clientes

Para el caso de los computadores clientes, estos deben tener los siguientes requerimientos de hardware:

- Computador Pentium II de 400 Mhz como mínimo.
- Espacio en Disco Duro de 1 Gb como mínimo
- Memoria RAM de 64 Mb como mínimo.
- Tarjeta de red para la interconexión local.

Los computadores clientes necesitan los siguientes requerimientos de software:

- Sistema operativo Windows 9X, ME, XP, NT o 2003
- Browser para navegación (Internet Explorer o Nestcape Navigator)

7.4.4 Pasos iniciales

Una vez que contamos con el servidor y se tiene ya instalado los requerimientos de software, debemos empezar a configurar nuestra aplicación. Para esto, en el directorio donde se instaló Mono, en la sección donde se

encuentra el directorio del servidor Web xsp, existe un archivo de configuración llamado **web.config**.

Este archivo contiene la información inicial para que la aplicación basada en Web funcione correctamente. En este archivo se debe indicar cual será el proveedor para la base de datos que estamos utilizando. Para nuestro caso, utilizamos la base de datos MySQL, por tanto el proveedor de base de datos que utilizaremos es **ByteFX.Data**, el cual fue estudiado en el Capítulo IV.

En este archivo se puede especificar además la cadena de conexión que se utilizará para conectarse a la base de datos, pero no es muy recomendable en especial si se tiene acceso a más de una base de datos, lo cual se soluciona indicando la cadena de conexión en el código fuente de las páginas que acceden a bases de datos.

La cadena de conexión que utilizamos se la específica en el archivo **funcion_tesis.inc**, la cual tiene es la siguiente:

```
cncString = "Server=ns.landetap.com;User ID=root;Password=root;Database=ccc";
```

Los datos de la cadena de conexión pueden variar dependiendo de la configuración del servidor donde se instale el sistema.

Para más información de cómo realizar la instalación y configuración del servidor para que el sistema este operativo, consulte la sección *Manuales* en los anexos de la tesis, los cuales se encuentran en el cd adjunto.

7.5 ESTRUCTURA FINAL DEL APLICATIVO

7.5.1 Módulo Informativo

- Contiene toda la información concerniente a los cursos y seminarios que se ofrecen o están disponibles en el Centro de Capacitación Continua (CCC).
- Presenta información de los cursos o seminarios con sus respectivos contenidos (temas, subtemas y horarios de clase);
- Presenta información acerca de los eventos por realizarse en la FICA.
- Presenta información acerca de los datos personales de los instructores con su respectiva hoja de vida (experiencia adquirida y la capacitación recibida).
- Presenta información acerca de los alumnos inscritos en un determinado curso o seminario con información acerca de los pagos realizados para un curso o seminario específico y sus respectivas notas recibidas.
- Presenta información acerca de las múltiples sugerencias y comentarios que los alumnos hagan acerca de un curso o seminario recibido.

7.5.2 Módulo de Inscripciones

Las personas interesadas en un curso o seminario, tienen la posibilidad de inscribirse en uno o más cursos o seminarios que se ofrezcan.

- Ingreso de los datos personales
- Posibilidad de acceso de ese usuario a través de un nombre de usuario y contraseña para revisar las notas e ingresar sugerencias y comentarios.

7.5.3 Módulo Administrativo

Es importante la administración del sitio Web, y se tiene la posibilidad de acceder al módulo administrativo mediante el ingreso de un nombre de usuario y contraseña para:

- Sacar reportes de Cursos, Seminarios, Instructores y Áreas.
- Sacar reportes de Alumnos inscritos en un determinado curso o seminario
- Ingresar nuevos cursos, seminarios, instructores y áreas
- Inscribir a nuevos alumnos
- Ingresar los pagos de un curso o seminario.
- Eliminar cursos o seminarios

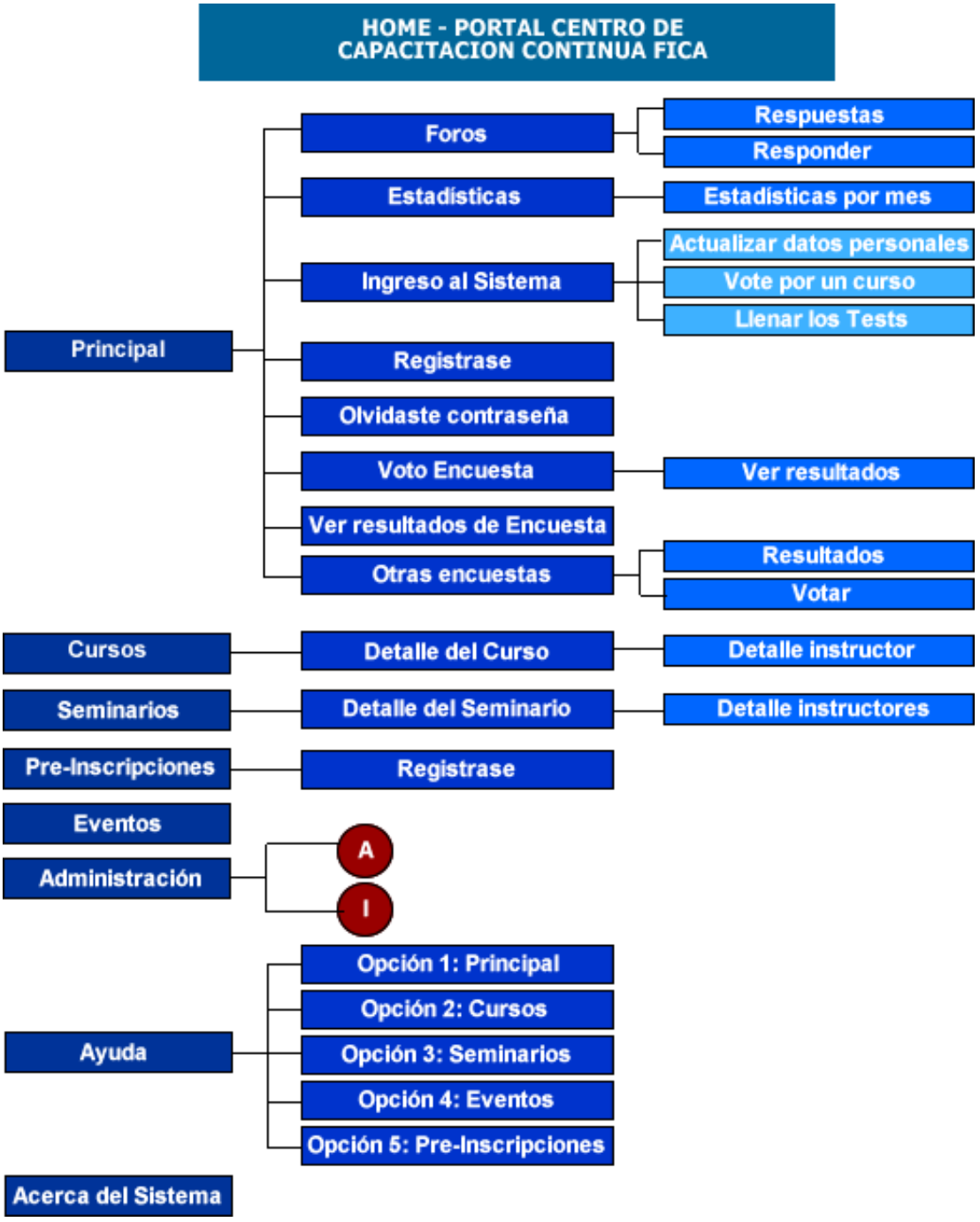
Sacar reportes históricos de cursos o seminarios en un determinado período y los respectivos alumnos que recibieron esa capacitación.

7.6 MAPA DEL SITIO

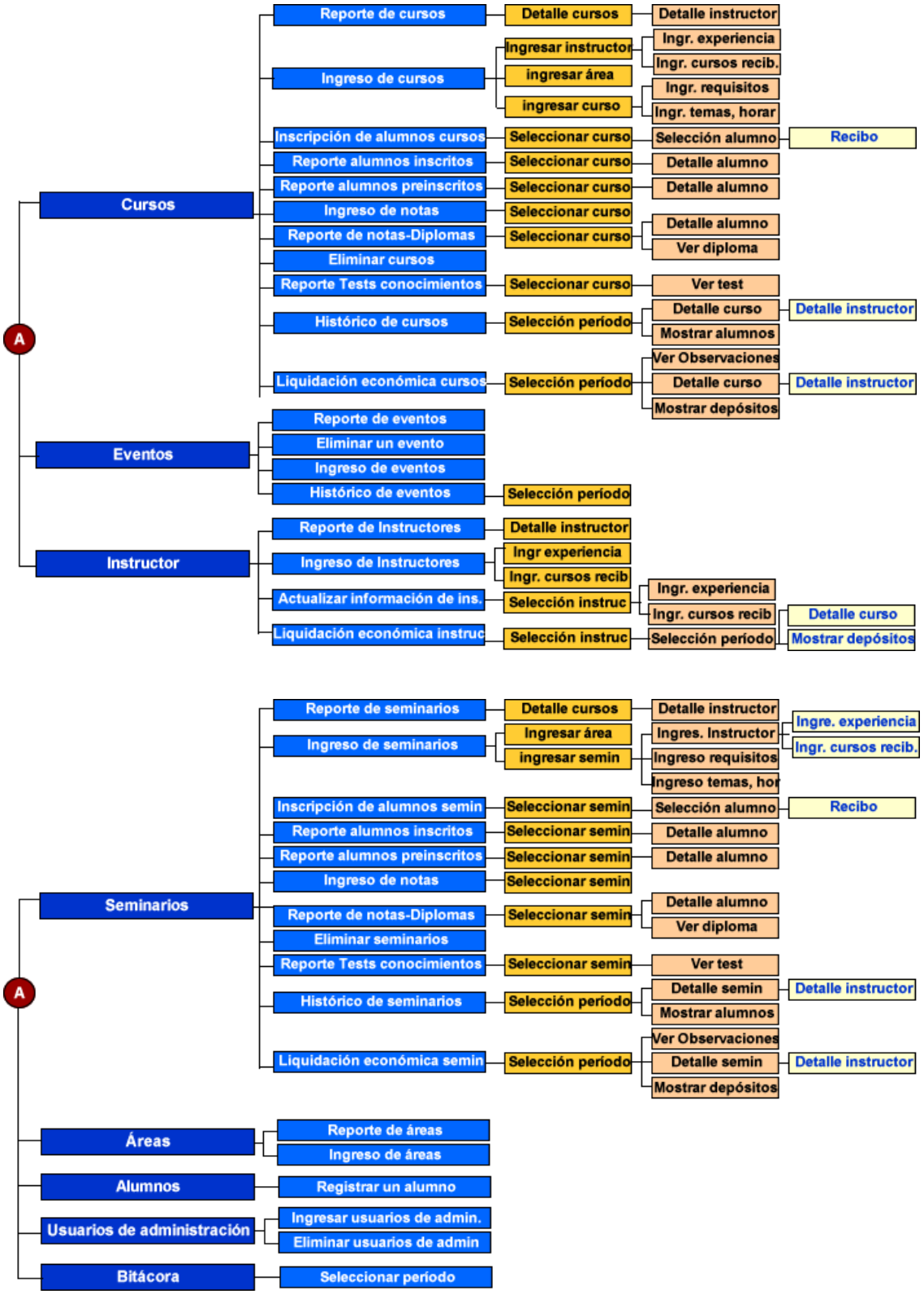
Una vez concluido el aplicativo, es necesario tener un Mapa del Sitio, el cual muestra de una manera gráfica los diferentes vínculos que se tienen en el Portal; de esta manera es posible definir e identificar los saltos que se realizarán para avanzar en el portal desde un punto A a un punto B.

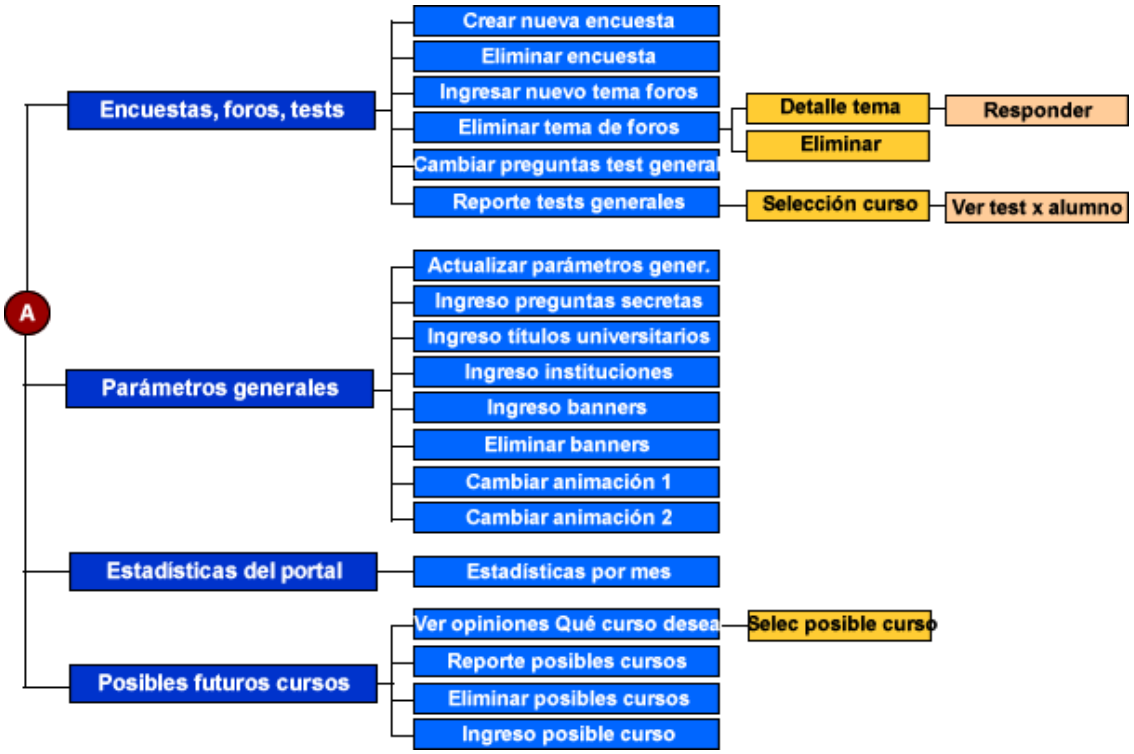
Este mapa incluye los gráficos para los tres tipos de usuarios que tiene el portal: alumnos, instructores y administradores.

MAPA DEL SITIO – AUTENTICADOS COMO ALUMNOS Y NO AUTENTICADOS

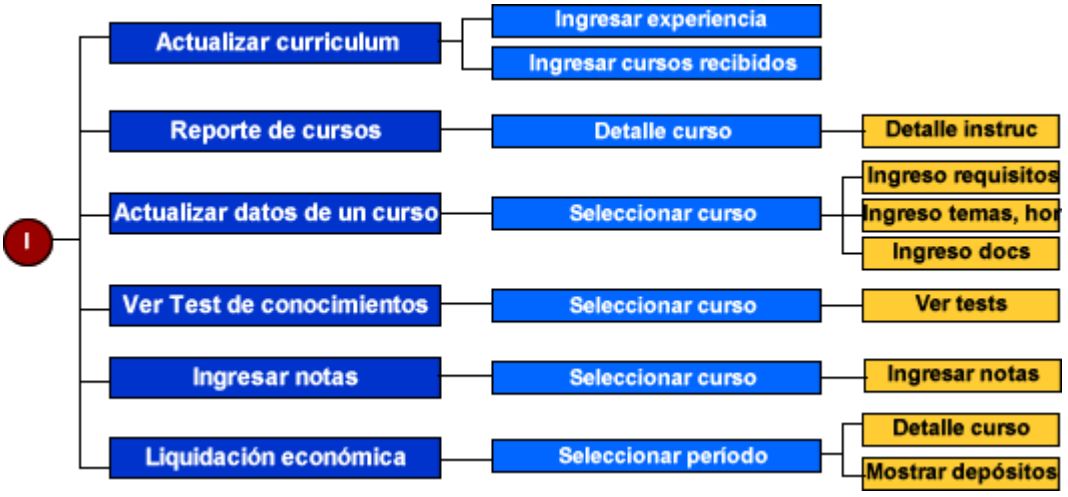


MAPA DEL SITIO – AUTENTICADOS COMO ADMINISTRADORES





MAPA DEL SITIO – AUTENTICADOS COMO INSTRUCTORES




CAPÍTULO VIII
VERIFICACIÓN DE HIPÓTESIS, CONCLUSIONES Y RECOMENDACIONES
Verificación de Hipótesis
Conclusiones
Recomendaciones

8.1 VERIFICACIÓN DE HIPÓTESIS

La hipótesis de la presente investigación fue:

Las aplicaciones generadas en la plataforma Linux con el proyecto Mono se podrán ejecutar sin problemas en la plataforma Windows. Esto es un factor determinante que ayudará al avance del desarrollo de software de código abierto.

Para verificar la hipótesis, se instaló el siguiente software en el sistema operativo Windows XP:

- .NET Framework 1.1
- Microsoft Data Access Components (MDAC) 2.7[43], el cual es un software para conectividad con bases de datos desde Windows utilizando .NET.
- MySQL Server 3.23 (version para Windows)[44]
- Proveedor nativo de MySQL (ByteFX.Data 7.4) para manejar .NET[45], es decir que es un driver de MySQl para tecnología .NET
- Microsoft ASP.NET Web Matrix Project[46], el cual permite diseñar páginas web ASP.NET en Windows

Una vez instalados estos componentes, se creo la base de datos en MySQL para Windows y se trasladaron las páginas ASP.NET desde el servidor Linux hacia el servidor Windows.

[43] MDAC 2.7: http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=B41304CA-874F-421D-8820-182F179779A4
[44] MySQL server: http://www.mysql.com
[45] ByteFX.Data 7.4: http://www.mysql.com/articles/dotnet/
[46] Microsoft ASP.NET Web Matrix Project: http://www.asp.net/webmatrix/guidedtour/getstarted/intro.aspx

Para ejecutar estas páginas, se utilizó Microsoft ASP.NET Web Matrix Project, el cual incluye un pequeño servidor Web para ejecutar las páginas ASP.NET.

Se ejecutaron las páginas ASP.Net y se llega a la conclusión de que **la Hipótesis no se cumple**

Esto debido a las siguientes razones muy importantes:

- En la plataforma Windows con la tecnología .NET de Microsoft no permite usar múltiples etiquetas **</form>**[47] que corren en el servidor (**runat="server"**). Bajo esta plataforma, se pueden usar múltiples etiquetas **</form>** pero con la restricción de que en tiempo de ejecución únicamente una este visible.

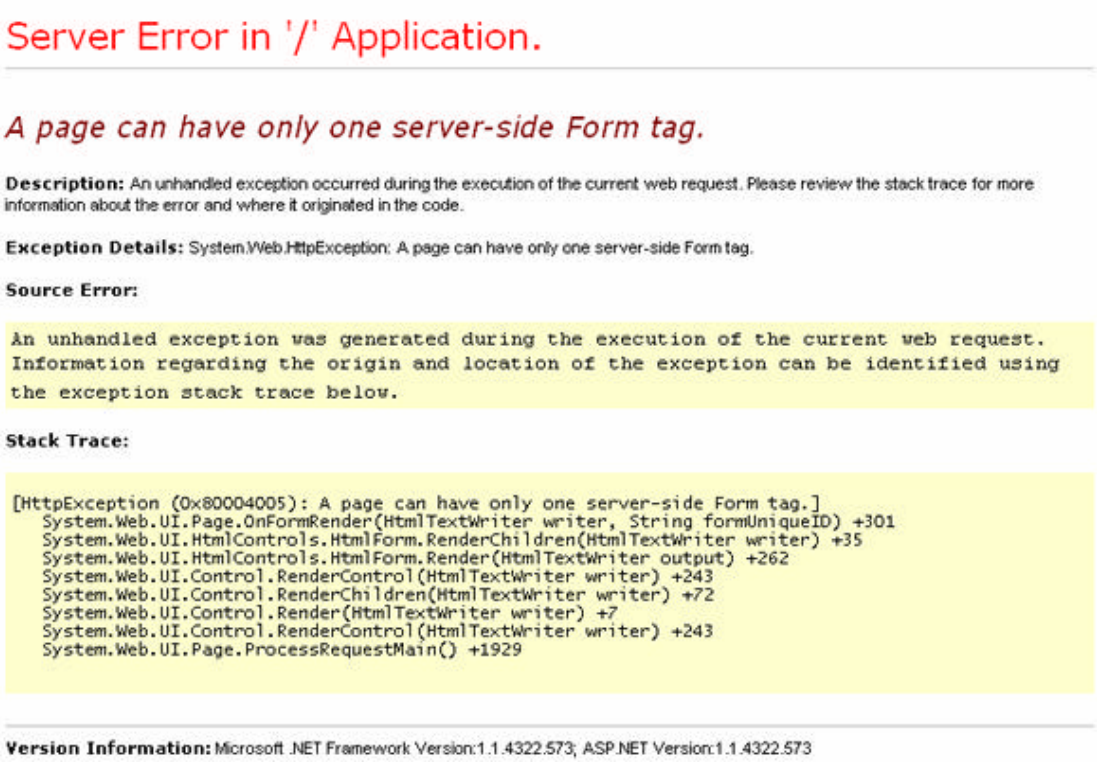
  En la plataforma Linux con la implementación de .NET de Mono, se pueden utilizar múltiples etiquetas **</form>** sin ninguna restricción y pueden estar Visibles dos o más al mismo tiempo.

  En el caso del Portal para el Centro de Capacitación Continua de la FICA se mantienen visibles dos etiquetas **</form>** simultáneamente, situación que es permitida por Mono y denegada por Microsoft .NET.

  La aplicación realizada en Linux con Mono funcionaría correctamente en Windows con Microsoft .NET realizando modificaciones acerca de las etiquetas **</form>** en el código fuente de todas las páginas Web.

  El mensaje de error que se obtuvo es el siguiente:

[47] Programación basada en formularios en ASP.NET: http://msdn.microsoft.com/msdnmag/issues/03/05/CuttingEdge/default.aspx



Server Error in '/' Application.

A page can have only one server-side Form tag.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Web.HttpException: A page can have only one server-side Form tag.

Source Error:

```
An unhandled exception was generated during the execution of the current web request.
Information regarding the origin and location of the exception can be identified using
the exception stack trace below.
```

Stack Trace:

```
[HttpException (0x80004005): A page can have only one server-side Form tag.]
   System.Web.UI.Page.OnFormRender(HtmlTextWriter writer, String formUniqueID) +301
   System.Web.UI.HtmlControls.HtmlForm.RenderChildren(HtmlTextWriter writer) +35
   System.Web.UI.HtmlControls.HtmlForm.Render(HtmlTextWriter output) +262
   System.Web.UI.Control.RenderControl(HtmlTextWriter writer) +243
   System.Web.UI.Control.RenderChildren(HtmlTextWriter writer) +72
   System.Web.UI.Control.Render(HtmlTextWriter writer) +7
   System.Web.UI.Control.RenderControl(HtmlTextWriter writer) +243
   System.Web.UI.Page.ProcessRequestMain() +1929
```

Version Information: Microsoft .NET Framework Version:1.1.4322.573; ASP.NET Version:1.1.4322.573

? En la plataforma Windows con la tecnología .Net de Microsoft es muy estricto en cuanto a que si se abre una conexión para base de datos, esta debe cerrarse necesariamente para poder abrir otra. Esto limita el uso de conexiones abiertas de forma anidada.

En la plataforma Linux con la implementación de .Net de Mono, no restringe el uso de conexiones a base de datos, y cuando no se cierra una conexión no da ningún tipo de conflictos. Por esta razón es posible usar conexiones abiertas de forma anidada.

En el caso del Portal para el Centro de Capacitación Continua de la FICA se utiliza conexiones a base de datos anidadas y en Linux no da ningún tipo de conflictos, cosa que no sucede en Windows donde se aplican restricciones a esta situación.

La aplicación realizada en Linux con Mono funcionaría correctamente en Windows con Microsoft .NET realizando cambios en la estructura de todas las funciones que utilicen conexiones anidadas a la base de datos, lo que implicaría el replanteo del código fuente y su estructura de programación.

Estos casos hacen que, para que se pueda realizar una aplicación en Linux con Mono y ejecutarla en Windows con .NET, se tenga que cambiar la estructura de las páginas Web así como también la lógica de la programación, lo cual dificulta el trabajo para el programador.

8.2 CONCLUSIONES

Luego de haber realizado la investigación de la implementación de .Net para la plataforma de desarrollo libre Linux, se tiene las siguientes conclusiones:

- .Net es una herramienta de desarrollo independiente de la plataforma, esto debido a que no compila a código de máquina, sino que compila a código intermedio lo que lo hace Multiplataforma.

- .Net es una herramienta de desarrollo independiente del lenguaje. Se puede usar una variedad de lenguajes para el desarrollo como son C#, Visual Basic y Java lo que permite que el programador desarrolle con el lenguaje que mejor se adapte.

- .Net tiene una gran variedad de librerías de clase que permiten: soporte para XML, soporte para algunos tipos de bases de datos, soporte para servicios Web XML y soporte para implementación de páginas Web dinámicas.

- Mono es una herramienta muy poderosa para la creación de software, es un gran proyecto que todavía no culmina pero que integra a muchas personas alrededor del mundo y que trabajan para cubrir todas las necesidades de la nueva tecnología.

- Mono es mucho más flexible que Microsoft .NET, ya que nos permite utilizar una estructura de programación en donde se puede tener abiertas algunas conexiones a la base de datos de forma simultanea y no causa problemas. Además se pueden mantener visibles algunas etiquetas </form> que se ejecutan en el servidor de forma simultanea y sin causar errores en la ejecución de las páginas Web.

- Una gran desventaja de Mono es que no cuenta con un entorno de trabajo (IDE – Integrated Development Environment), lo cual obliga a programar en cualquier editor de texto y no de forma gráfica como las múltiples herramientas de trabajo que utiliza Microsoft .NET como es el caso de Visual Studio y Microsoft ASP.NET Web Matrix.

- El sistema operativo Linux, es muy confiable en el momento que debemos utilizar un servidor Web y de base de datos, demostrando toda su potencialidad en aplicaciones como la que hemos construido como parte de este trabajo de investigación.

- MySQL es una base de datos muy liviana y también muy segura al momento de necesitar gran fiabilidad en los datos que se mantienen guardados.

- El desarrollo del software libre es una actividad que cada día gana muchos más adeptos, ya que no se debe desaprovechar todas las ventajas que se tiene al trabajar con una licencia como la GPL, en donde no se tiene las restricciones que se tiene con software que se adquiere muchas veces a altas sumas de dinero.

- El aplicativo desarrollado puede ser de gran utilidad para los estudiantes de la FICA y del público en general, ya que ayudará a optimizar los procesos que se llevan a cabo en la actualidad en lo referente a cursos y seminarios.

- El aplicativo desarrollado ha demostrado en la práctica la gran funcionalidad de Mono en lo referente a aplicaciones basadas en Web, inclusive teniendo en cuenta que Mono es una herramienta todavía en desarrollo

- El hecho de que Mono esté todavía en desarrollo no fue impedimento para el desarrollo del presente proyecto; se encontraron muchos obstáculos técnicos, pero estos fueron solucionados con soluciones informáticas que han sido aprendidas a lo largo de los años de carrera en la Universidad.

- Debido a que la versión de Mono (0,28) no permitía todavía el envío de archivos en formularios, se tuvo que recurrir al antiguo lenguaje de programación PHP para solucionar este problema. Esto se realizó para subir archivos al servidor (upload) y de esta manera permitir el envío de imágenes y otros documentos que serán mucha ayuda para los usuarios del sistema.

- Mono es una tecnología que sigue en un crecimiento constante, un ejemplo de esto es que cumplen estrictamente un cronograma establecido de generación de actualizaciones de los diferentes componentes de Mono, lo que ha permitido a los desarrolladores la solución de los múltiples problemas técnicos que han sufrido en el desarrollo de sus aplicaciones.

8.3 RECOMENDACIONES

- Se recomienda continuar con el estudio de los Servicios Web XML como un futuro tema de tesis, ya que es una tecnología muy innovadora y que cubre una amplia gama de posibilidades en el desarrollo de aplicaciones Web distribuidas.

- La presente investigación cubre el estudio de Mono utilizando aplicaciones ASP.NET, pero además existe otro tipo de aplicaciones basadas en ventanas (Windows Forms) que no se han estudiado en el presente trabajo investigativo y que se pueden aplicar para desarrollar software. Se recomienda la investigación de Mono junto a la plataforma Linux utilizando herramientas como GTK# o Glade, con lo cual se puede generar aplicaciones basadas en ventanas.

- Además se recomienda extender el estudio de .NET utilizando Mono para otras áreas de programación como son la comunicación remota (remoting) y conexiones de red (networking), aspectos que pueden ser muy interesantes para solucionar problemas actuales de interconexión o simplemente mejorar las condiciones actuales.

- Se recomienda incentivar a los estudiantes de la Escuela de Ingeniería en Sistemas Computacionales para que estudien las tecnologías que se muestran en el presente trabajo de investigación de manera que se de un aporte al desarrollo del software libre y un aporte a las nuevas tecnologías.

- Se recomienda además, conocer el estado actual del Portal del Centro de Capacitación Continua, ya que como cualquier sistema que se desarrolla, necesita mantenimiento y actualización para en un ciclo repetitivo de manera que no se vuelva obsoleto y que sea siempre siga el objetivo para el que fue diseñado.

- Se recomienda fomentar en los estudiantes de la Escuela de Ingeniería en Sistemas Computacionales para que utilicen el sistema operativo Linux, y descubran todas sus ventajas frente a otros sistemas operativos para servidor.

- Es muy importante no perder de vista al proyecto Mono que ejecuta la empresa Ximian, ya que es un proceso que no ha terminado aún, e incluso se puede aportar con la creación de muchos componentes y librerías que todavía no se han creado, pasando a formar parte de toda una legión de programadores que apoyan al desarrollo de herramientas de código abierto.

- Es indispensable conocer los avances que va teniendo Microsoft .Net en comparación con lo que Mono continua incrementando, ya que lo que el equipo de desarrolladores de Ximian y Novell quiere es estar a la par con Microsoft en la innovación tecnológica de la tecnología .Net.

- El aplicativo desarrollado se lo realizó en la versión 0,28 de Mono. El equipo de desarrolladores sigue actualizando las herramientas, de esta manera se tiene ya la versión 1.0 de Mono. Se recomienda la investigación de las nuevas actualizaciones, ya que en cada nueva actualización se integran nuevas características y posibilidades.

BIBLIOGRAFÍA

LIBROS

- ARCHER,Tom, “A Fondo C#”, Editorial Mc Graw-Hill, 2001
- RODRIGUEZ, Miguel, “Desarrollo de Aplicaciones .NET con Visual C#”, Editorial Mc Graw-Hill, 2002
- BÜHLER, Erich “Visual Basic .NET”, Editorial Mc Graw Hill, 2002
- YOUNG, Michael “Aprenda XML Ya”, Editorial Mc Graw Hill, 2000
- DUBOIS, Paul, “MySQL”, Editorial Prantice Hall,2001
- FONSECA, Gustavo, “E-Commerce con Linux”, Editorial Prantice Hall, 2001
- DEGLER, Stephen, “Administración de Sistemas Linux”, Editorial Prantice may, 2000
- LEBLANC, Dee-Ann, “Administración de Sistemas Linux”, Editorial Anaya Multimedia, 2001

REFERENCIAS WWW

- http://www.go-mono.com/
- http://www.informit.com/isapi/product_id~{E677E33F-8FA6-4543-96E4-259920DA7106}/element_id~{55D2B30E-1C1A-4B95-88EA-ACBE727F5613}/st~{EA7C8D03-4995-402D-B085-06E000F897B8}/content/articlex.asp
- http://www.clikear.com/manual_csharp/c53.asp
- http://www.clikear.com/manual_csharp/c10.asp
- http://codorniz.arcos.inf.uc3m.es/msaa/documentos/Como2.html
- http://www.asp.net/Tutorials/quickstart.aspx
- http://www.asp101.com/articles/matt/securesite/default.asp
- http://www.webreference.com/js/tips/021020.html
- http://www.codeproject.com/aspnet/webforms.asp?print=true
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconadonetarchitecture.asp

? http://longhorn.msdn.microsoft.com/lhsdk/ndp/cpconadonetarchitecture.aspx
? http://www.dnjonline.com/articles/essentials/iss22_essentials.html
? http://msdn.microsoft.com/msdnmag/issues/1100/adoplus/default.aspx
? http://es.gotdotnet.com/quickstart/howto/doc/adoplus/ADOPlusOverview.aspx
? http://www.alistapart.com/articles/webservices/
? http://msdn.microsoft.com/webservices/understanding/
? http://www.sys-con.com/webservicesedge2003east/86.cfm
? http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/index.html


ANEXOS
ANTEPROYECTO DE TESIS
Tema
Problema
Justificación
Objetivos
Alcance del Aplicativo
Marco Teórico
Hipótesis
Metodología
Tabla de Contenidos
Costos
Bibliografía
Cronograma de Actividades

TEMA

Implementación de Aplicaciones .NET para plataforma de desarrollo libre, basada en Linux y compatible con Microsoft.NET.

Aplicativo: Portal para el Centro de Capacitación Continua de la FICA.

PROBLEMA

Para los desarrolladores ha sido siempre un impedimento el no poder hacer aplicaciones que funcionen en cualquier plataforma. Ha sido una imposición más que una necesidad el tener que desarrollar aplicaciones en la plataforma comercial Windows; y por otro lado se tiene a la plataforma de desarrollo libre Linux con la robustez y efectividad que ha demostrado a través de los años.

Otro problema ha sido siempre la dependencia del lenguaje, obligándonos a aprender uno y otro lenguaje dependiendo de la plataforma y la herramienta que se utilice para desarrollar aplicaciones.

JUSTIFICACIÓN

Ximian es una empresa dedicada al Software libre, más comúnmente llamado software de “código abierto”, como una contribución al proyecto GNOME y otros esfuerzos de la comunidad de desarrolladores de software de código abierto. Ximian hace bastante por el desarrollo de código abierto bajo los términos de la General Public License (GPL) y Lesser General Public License (LGPL).

El Proyecto Mono es una iniciativa de la comunidad para desarrollar una versión de la plataforma de desarrollo Microsoft.NET que sea de código abierto y basada en Linux. El punto clave es añadir el lenguaje C# al arsenal de herramientas de desarrollo de código abierto y permitir la creación de programas .NET operativos e independientes.

El Proyecto Mono provee:

- Un compilador C# extendiendo la plataforma de desarrollo GNOME o KDE que permite a los desarrolladores de Linux crear aplicaciones compatibles con .NET.
- Una completa implementación de librerías de dase compatibles con el Microsoft CLI (Common Language Infrastructure), habilitando a los desarrolladores a crear aplicaciones para el usuario final como pueden ser servicios web usando la funcionalidad de bases de datos accesible en los sistemas de código abierto.
- Una versión para Linux del Microsoft Common Language Run Time (CLR) y del compilador Just-in-time (JIT) que permite a los sistemas Linux correr aplicaciones .NET construidas en las plataformas Windows, Linux o Unix.

Microsoft.NET es una iniciativa muy grande para proveer herramientas de desarrollo tanto para aplicaciones GUI como para Servicios Web (desarrollo .NET Framework). Las herramientas de desarrollo .NET, las cuales incluyen al compilador C# y CLI (Common Language Infrastructure), permiten a programas escritos en C# y otros lenguajes correr en sistemas operativos diferentes a Windows. Microsoft y Corel anunciaron recientemente que van a colaborar en la realización de la plataforma FreeBSD bajo los términos de la licencia de “código libre” de Microsoft.

Mientras C# y CLI (Common Language Infrastructure) han sido incluidas en los estándares ECMA (European Computer Manufactures Association) y W3C (World Wide Web Consortium), las limitaciones de las licencias del “código libre” de Microsoft pueden impedir el desarrollo y construcción de aplicaciones comerciales sobre otras plataformas, como puede ser Linux, Solaris y Unix.

En contraste con esto, el proyecto Mono como plataforma de desarrollo, provee a los desarrolladores de código abierto lo que se diría: “constrúyelo una

vez, utilízalo donde sea" tomando ventaja con esto a los servicios habilitados por Microsoft.NET. Bajo los términos de las licencias GPL (General Public License) y de LGPL (Lesser General Public License) usadas por el proyecto Mono, los desarrolladores pueden escribir y distribuir aplicaciones comerciales y propietarias, algo que no es posible con la licencia de "código libre" de Microsoft.

El proyecto Mono provee tres elementos clave en una estructura de desarrollo designada a permitir a los programadores una rápida creación, uso y corrida de aplicaciones compatibles con .NET sobre la plataforma Linux. Un compilador C# extendiendo la plataforma de programación GNOME permite a los desarrolladores de Linux crear aplicaciones compatibles con .NET. Aquellos programas se pueden construir sobre una completa implementación de librerías de clase compatibles con el CLI (Common Language Infrastructure) de Microsoft, habilitando a los programadores a crear aplicaciones de usuario final como pueden ser poderosos servicios web usando la funcionalidad de una base de datos permitida en los sistemas de código abierto. Por último, una versión Linux de Microsoft CLR (Common Language Run Time) y JIT (Just In Time) permite a los sistemas Linux correr aplicaciones .NET construidas en Windows, Linux o Unix.

La plataforma de desarrollo .NET es muy rica, poderosa y es una plataforma bien diseñada, tanto que puede ayudar al desarrollo en la plataforma de software libre. Hay que recordar que, así como el proyecto GNU inició clonando a Unix hace diez y seis años atrás, el proyecto Mono está clonando la plataforma de desarrollo .NET porque es una gran plataforma para construir sobre ella.

OBJETIVOS

Generales:

- Realizar un estudio de la programación .NET basada en la plataforma de desarrollo libre Linux
- Generar una aplicativo de entorno Web que permita mostrar la información referente al Centro de Capacitación Continua de la FICA.

Específicos:

- Hacer un estudio de las diferentes posibilidades que ofrece .NET
- Hacer un estudio de aplicaciones basadas en Windows (Windows Forms)
- Hacer un estudio de aplicaciones basadas en Web (Web Forms)
- Hacer un estudio de los servicios Web XML.
- Hacer un estudio de la integración .NET y Bases de Datos

ALCANCE DEL APLICATIVO

Módulo Informativo

Contendrá toda la información concerniente a los cursos y seminarios que se ofrecen o están disponibles en el Centro de Capacitación Continua (CCC).

Así se tendrá:

- Capacitación Básica
- Actualización Profesional
- Seminarios Especiales

Por cada uno de los cursos o seminarios se tendrá:

- Título del curso o seminario
- Objetivos del curso o seminario
- Programación: extensión, fechas, tutores
- Contenidos
- Información de Inscripciones.

Módulo de Inscripciones

Las personas interesadas en un curso, tendrán la posibilidad de inscribirse en uno o más cursos que se ofrezcan.

- Ingreso de los datos personales
- Posibilidad de acceso de ese usuario a través de un nombre de usuario y contraseña
- Posibilidad de actualización de esos datos.

Módulo Académico

Es necesario llevar un control de las personas inscritas para los diferentes cursos y seminarios. Una vez que ha empezado un curso es necesario llevar un control de notas para una posterior aprobación o no del curso.

- Posibilidad de acceder a reportes de notas.
- Acceso a reportes de aprobación o desaprobación de cursos.

Control Estadístico

En base a los resultados obtenidos es posible llevar un control estadístico del portal, así es posible obtener estadísticas del número de cursos, número de inscritos, etc.

MARCO TEÓRICO

Qué es .NET?

.NET es una familia de productos, lanzado por Microsoft en el 2001, como una respuesta al éxito que tuvo Java de Sun. No es un clon, pero es un completo rediseño, con unas nuevas excitantes características, como la independencia del lenguaje, gran soporte para bases de datos, soporte XML, buen diseño de Servicios Web y Aplicaciones Web, soporte de una gran actualización del lenguaje Visual Basic, incluyendo orientación a objetos. A esta familia también se añade los servidores de Microsoft.

Microsoft .NET es de hecho 3 cosas. Es importante no confundirlas:

- El .NET Framework
- Serivicios como MyServices o Passport
- Otras cosas con la etiqueta ".NET"

Entonces nace Mono, que es solo una implementación del .NET Framework en Linux. El .NET Framework es una “plataforma de software”, similar a Java.

La independencia del Lenguaje. Quizás el factor más importante es, que todos los lenguajes “son creados evaluadamente”. No hay uno o dos lenguajes, que pueden ser consumidos por otros, si se quiere ser independiente del lenguaje. Por ejemplo, todos los APIs Gnome son escritos en C. Esto asegura que tu puedes escribir una aplicación en Python, por el acceso a las librerías C a través de librerías. Pero no se puede acceder a los programas Python a través de C. En el mundo .NET esto es posible. Escrito en C#, MonoBasic, o lo que sea, tu puedes sin ningún problema reutilizar esas clases.

Independencia de la Plataforma. El código .NET no es compilado a código de máquina, pero hay una forma intermedia. Se puede copiar binarios a través de las plataformas como en Java. Únicamente la invocación del código nativo puede parar esto, pero desaparecerá con el tiempo.

Librerías de Clase. .NET tiene una gran librería de clases muy grande, especialmente las clases de alto nivel son interesantes: Soporte XML, soporte de Base de Datos, soporte de Servicios Web y Paginas Web son algunos ejemplos.

Lenguaje C#. C# es un lenguaje moderno, simple y orientado a objetos. Es muy parecido a C y C++ y combina la alta productividad de los lenguajes RAD (Rapid Application Development) y el poder de C++.

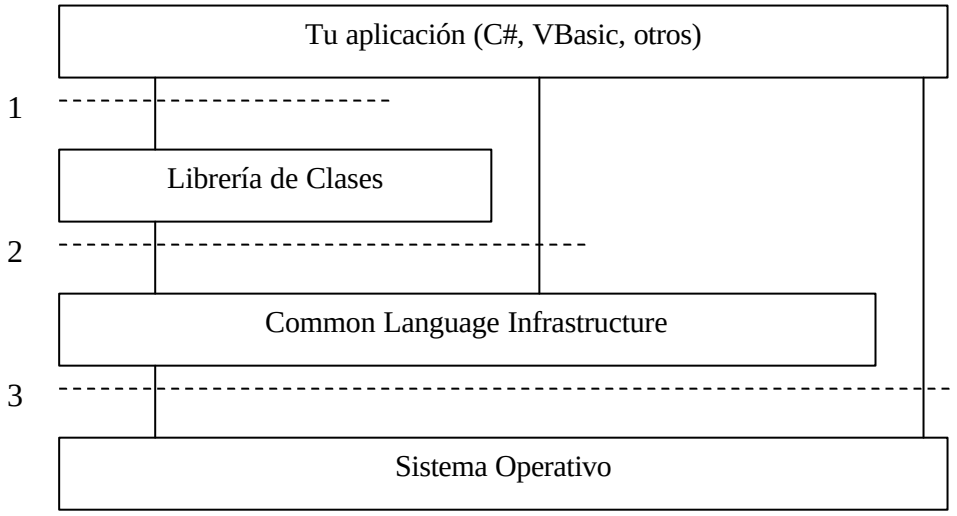
VB.NET. Microsoft finalmente fijó el lenguaje Basic aquí. Ahora tiene muchas nuevas características como C# o cualquier otro. En .NET las clases Basic pueden ser rehusadas en C# y viceversa.

Aplicaciones Web. Con ASP.NET se embarca en la siguiente generación de la estructura de páginas Web. Esto surge de la independencia del lenguaje, así

que finalmente se puede escribir las páginas Web con Pascal o C++. También es la primera estructura en hacer a HTML accesible a objetos, mientras introduce también complejas cosas como calendarios.

Arquitectura .Net Framework

Cuando nos aproximamos a algo que es tan complejo como .NET, nos ayuda a darnos una idea de cómo se relaciona esto con nuestro trabajo. El siguiente diagrama nos da un contexto de donde nuestra aplicación se ajusta dentro de Mono.



En el gráfico se puede ver que nuestra aplicación escrita en un lenguaje compatible con Mono interactúa directamente con las Librerías de Clases por medio de la inclusión de librerías (ejemplo: usando “System”). La aplicación interactúa con el Common Language Infrastructure por medio de las especificaciones CLI (Common Languages Specification) y con el sistema Operativo por medio de los API’s nativos el Sistema Operativo.

Librerías de Clase.

La librería de clases provee un comprensible conjunto de facilidades para el desarrollo de aplicaciones. Primeramente es escrito en C#, y puede ser usado por cualquier lenguaje gracias al Common Languaje Specification .

La librería de clases es estructurada en Namespaces, y usada en las librerías libres conocidas como Ensamblajes.

Cuando hablamos del .Net Framework, nos referimos a esta librería de clases.

Namespaces

Namespaces es un mecanismo para un agrupamiento lógico similar a las clases en una estructura jerárquica. Esto previene el conflicto de nombres. La estructura es implementada usando palabras separadas con punto. El nivel más alto de Namespaces para .Net Framework es System. Hay otros niveles altos de Namespaces como es Windows por ejemplo. Se puede crear los nuestros también.

Ensamblajes

Ensamblajes son el empacamiento físico de las librerías de clase. Aquí tenemos archivos .dll, pero no debemos confundirlos con las librerías Win32. Ejemplos: System.dll, Accessibility.dll, etc.

Common Languaje Infrastructure.

Más comúnmente llamada como Common Language Runtime, es implementada por el ejecutable de Mono. Este es usado para ejecutar la compilación de aplicaciones .NET. Este ha sido definido por el ECMA, en el estándar ECMA-335.

Common Languaje Specification

Está especificado en ECMA-335, capítulo 6. Define una interfaz para el CLI (Common Language Infrastructure), por ejemplo convenciones para los tipos de datos.

El proceso entonces es: El compilador Mono genera una imagen que lo conforma el CLS. Este es el Lenguaje Intermedio Comun (Common Intermediate Language). El runtime de Mono toma esta imagen y la corre.

Implementación Mono

Lenguaje C#

Mono introduce un nuevo y completo lenguaje, el cual es muy similar a Java, pero también tiene algunas características de C++.

Lenguaje VB

El lenguaje Basic no ha sido realmente cambiado a través de los años. Sin embargo VisualBasic.NET integra algunas características, Basic es ahora más poderoso como C# u otros lenguajes. Mono lo embarca en el compilador CIL (Common Language Infrastructure), haciendo al lenguaje tan digno como C#. Gracias a la independencia del lenguaje, las clases escritas en MonoBasic pueden ser usadas en C# muy bien. La carencia de la implementación de un buen soporte Basic en Unix ha sido finalmente realizada.

ADO.NET

Mono provee una interfaz común para Bases de datos., así que el cambio será muy fácil. Difícilmente se encontrará una base de datos no soportada, ya sea directamente o a través de ODBC. Es muy portable y trabaja en todas las plataformas.

ASP.NET

ASP.NET es una gran estructura para desarrollar sitios Web y servicios Web. Es un lenguaje independiente también. Imaginemos una página Web escrita en LOGO, Basic, C#, C++, etc. Se puede acceder a todas las características de las librerías de clase, y es muy rápido porque todas las librerías son compiladas. ASP.Net también integra. WebForms, que hace a los elementos HTML accesibles como objetos.

GTK#

GTK (GIMP Toolkit) es una librería para la creación de interfaces de usuario gráficas. Tiene una licencia de LGPL (Lesser General Public License), así que

se puede desarrollar software abierto, software libre, o software comercial no libre usando GTK sin la necesidad de otras licencias.

XML

Puede ser usado como un archivo de datos o como un archivo con formato de documento, por ejemplo los formatos de HTML u Office. La ventaja es que hay muchas herramientas y API's accesibles trabajar con XML y pueden ser fácilmente editados manualmente.

HIPÓTESIS

Las aplicaciones generadas en la plataforma Linux con el proyecto Mono se podrán ejecutar sin problemas en la plataforma Windows. Esto es un factor determinante que ayudará al avance del desarrollo de software de código abierto.

METODOLOGÍA

Obtención de Información

Buscar la información necesaria de diferentes fuentes bibliográficas incluyendo el Internet como medio fundamental para encontrar información.

Estudio de Linux

Realización de un estudio preliminar de este sistema operativo de código abierto, incluyendo la instalación y utilización del mismo.

Estudio de Microsoft.NET

Realización de un estudio preliminar de las características de Microsoft.NET como una etapa inicial para el futuro estudio de .NET para Linux: Mono.

Estudio de Mono

Una vez que se conoce las características de .NET es momento para empezar a conocer Mono, incluyendo su instalación y sus primeros pasos de utilización.

Investigación del alcance del aplicativo

Realizar el análisis de los requerimientos del Centro de Capacitación Continua para la realización del aplicativo.

Estudio de requerimientos tecnológicos para que el aplicativo funcione

Una vez realizado el análisis del alcance del aplicativo, se puede determinar los requerimientos tanto de hardware como de la red para que el aplicativo sea funcional.

Prototipo del aplicativo (Alfa) y pruebas necesarias

Se empezará realizando un prototipo denominado Alfa que será la primera versión del sistema, y por supuesto que se realizaran todas las pruebas necesarias

Prototipo del aplicativo (Beta) y pruebas necesarias

Luego se realizará la versión Beta del aplicativo con las correcciones necesarias de la versión Alfa y todos los ajustes que sean necesarios para que el aplicativo sea cien por ciento funcional.

Documentación de todo el aplicativo

Se realizará a lo largo de la realización del aplicativo

Conclusiones y recomendaciones

Después de realizar las pruebas, se determinará si se cumple o no las hipótesis planteadas y se dará conclusiones al respecto.

TABLA DE CONTENIDOS

CAPÍTULO VI

ASP.NET

6.1 ASP.NET

6.1.1 Web Forms

6.2 Servidores Web

6.2.1 XSP

6.2.2 Apache

6.3 Servicios Web

6.3.1 Descripción de servicios WEB (WSDL)

6.3.2 SOAP

CAPÍTULO VII

APLICATIVO: Portal del Centro de Capacitación Continua

7.1 Alcance

7.2 Plataforma

7.3 Análisis y Desarrollo del Aplicativo

7.4 Versión Alfa del Aplicativo

7.5 Versión Beta

7.6 Documentación

7.7 Pruebas Finales

CAPÍTULO VIII

CONCLUSIONES Y RECOMENDACIONES

8.1 Verificación de Hipótesis

8.2 Conclusiones

8.3 Recomendaciones

BIBLIOGRAFÍA

ANEXOS

Guía de Usuario de .NET

Instalador de Mono y librerías necesarias.

COSTOS

	Cantidad	Detalle	Costo Actual (USD)	Costo Real (USD)
Hardware	1	Computador PII 266 Mhz, 30 Gb, 160 Mb RAM	400	0
	1	Procesador Pentium IV 1,8 Ghz	150	150
	1	Mainboard Intel	80	80
	1	Impresora Epson Stylus Color	80	0
Software	1	Linux 8.0	10	0
	1	Mono	0	0
Varios	8	Cuentas mensual de Internet Plan hora	40	0
	4	Libros	300	300
	5	Papelería	50	50
	4	Tinta	100	100
		Movilización	50	50
Subtotal			1850	740
15% Imprevistos				111
TOTAL				* 851

***** Financiamiento personal.

BIBLIOGRAFÍA

Libros

- BÜHLER, Erich “Visual Basic .NET”, Editorial Mc Graw Hill, 2002
- YOUNG, Michael “Aprenda XML Ya”, Editorial Mc Graw Hill, 2000

Referencias www

- http://msdn.microsoft.com/netframework/productinfo/overview/default.asp
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp
- http://msdn.microsoft.com/netframework/productinfo/features/default.asp
- http://www.asp.net/webmatrix/default.aspx?tabindex=4&tabid=46

- http://aspnetpro.com/features/2002/08/asp200208sm_f/asp200208sm_f.asp
- http://www.go-mono.com/
- http://www.ximian.com/about_us/open/
- http://developer.ximian.com/projects/mono/
- http://www.gotmono.com/
- http://www.monohispano.org/