Universidad de las Ciencias Informáticas

Facultad 8

"Estrategia de portabilidad para desplegar aplicaciones Web desarrolladas con la tecnología .NET de Microsoft en plataforma Linux, utilizando MONO"

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Deivis Leyva Velázquez

Tutor: Lic. Alèx David Saad Blanco

Julio de 2008

AGRADECIMIENTOS

A las primeras personas que quiero agradecer son a mi papá y a mi mamá, por siempre estar a mi lado ayudándome y dándome el apoyo necesario para salir adelante en la vida. A ellos le debo mi existencia y gracias a ellos he llegado hasta donde estoy.

A la Revolución, por haberme dado la oportunidad de estudiar en una escuela como la UCI, una oportunidad que sólo puede ser posible en una Revolución Socialista como la nuestra, donde la educación es gratuita y todos tenemos las mismas oportunidades.

A Lestón, por ser la persona que mas apoyo me ha dado en la universidad y estar siempre ahí aconsejándome y soportándome.

A Alexis, Rayner, Ariel, la gente de la vanguardia con la que he compartido gran parte de estos cinco años.

A mi tutor David, por darme el apoyo necesario durante la realización del Trabajo de Diploma.

A Yuna, por nunca haberme abandonado en los momentos difíciles y haberme ayudado a salir adelante.

A mis compañeros de aula por haber compartido a lo largo de estos cinco años de la carrera, largas madrugadas, estudio incansable, actividades, actos políticos, etc.

DEDICATORIA

A mis padres, por siempre darme apoyo y preocuparse por mi cada instante de mi vida. A toda mi familia en general.

A nuestro Comandante en Jefe Fidel, por haber tenido la visión tan grande en la construcción de una obra tan majestuosa como la Universidad de las Ciencias Informáticas.

A Lester, por darme su apoyo en todo momento y tener tanta paciencia conmigo.

A mis compañeros de aula de estos cinco años, en especial a Yuna.

A todas esas personas que han puesto su confianza en mí, y por las cuales me siento comprometido a no fallarles y dar todo de mí cada día.

RESUMEN

En el presente trabajo se define toda una estrategia a tener en cuenta para lograr desplegar aplicaciones Web desarrolladas con la tecnología .NET de Microsoft en plataformas Linux, utilizando el marco de trabajo de Mono. Se exponen las principales características que comprende la implementación de .NET desarrollada por Mono, describiendo su estado actual y perspectivas futuras.

Se describen las acciones a llevar a cabo para verificar la compatibilidad de las aplicaciones con Mono, las principales incompatibilidades que pueden ser encontradas y como solucionarlas. También se realizan las configuraciones necesarias para poder desplegar las aplicaciones en Linux a través del servidor Web Apache.

El principal objetivo de la estrategia definida es contar con una alternativa que permita que las aplicaciones Web desarrolladas en ASP.NET desde Windows comprendan un alto nivel de compatibilidad con Mono, permitiendo de esta forma realizar su despliegue en cualquier momento sobre plataforma Linux.

PALABRAS CLAVE

Compatibilidad, portabilidad, .NET, Mono, multiplataforma.

RESUMEN

En el presente trabajo se define toda una estrategia a tener en cuenta para lograr desplegar aplicaciones Web desarrolladas con la tecnología .NET de Microsoft en plataformas Linux, utilizando el marco de trabajo de Mono. Se exponen las principales características que comprende la implementación de .NET desarrollada por Mono, describiendo su estado actual y perspectivas futuras.

Se describen las acciones a llevar a cabo para verificar la compatibilidad de las aplicaciones con Mono, las principales incompatibilidades que pueden ser encontradas y como solucionarlas. También se realizan las configuraciones necesarias para poder desplegar las aplicaciones en Linux a través del servidor Web Apache.

El principal objetivo de la estrategia definida es contar con una alternativa que permita que las aplicaciones Web desarrolladas en ASP.NET desde Windows comprendan un alto nivel de compatibilidad con Mono, permitiendo de esta forma realizar su despliegue en cualquier momento sobre plataforma Linux.

PALABRAS CLAVE

Compatibilidad, portabilidad, .NET, Mono, multiplataforma.

ÍNDICE

INTRODUCCIÓN

Si analizamos el mundo del desarrollo de software descubriremos que el estado actual es bastante mejorable: la tecnología que estamos utilizando es la misma hace una década, pero cada vez se está complicando más, esto trae como consecuencia que la curva de aprendizaje para los nuevos desarrolladores cada día que pasa sea más dura. Por otra parte la complejidad de los sistemas operativos y el número de versiones que coexisten concurrentemente es mayor. Ante esta situación parece sensato realizar una revisión de las herramientas actuales y la arquitectura de desarrollo, en aras de lograr la mayor estabilidad y estandarización posible dentro de nuestra organización en cuanto a las herramientas de desarrollo que utilizamos y la tecnología para desplegar nuestras aplicaciones.

Los proyectos grandes de software deben de hacer frente a problemas que no existen en proyectos menores. Los programas que tienen un ciclo de vida largo deben hacer frente de distinta forma a la gestión de memoria que los programas pequeños. Llega un momento en el que te das cuenta que has perdido demasiado tiempo escribiendo destructores, solucionando un problema de memoria, utilizando funciones no seguras de bajo nivel, y que has implementado demasiadas listas enlazadas.

La plataforma .NET es un gran avance sobre todo en cuanto a productividad, de forma que, aunque Microsoft haya desarrollado estas tecnologías teniendo en mente servicios Web, su mayor beneficio es aumentar la productividad del programador. Otra característica que nos brinda .NET es la independencia de lenguaje, permitiendo escoger el lenguaje más apropiado según el problema con el que nos enfrentamos. Contando además con un Entorno Integrado de Desarrollo (IDE, *por sus siglas en inglés*) tan potente como Visual Studio, el cual complementa y solidifica las potencialidades de la plataforma .NET.

Ahora, por otra parte esta presente la plataforma de despliegue que utilizamos. Los beneficios derivados del uso del sistema operativo UNIX, y por lo tanto de Linux, provienen de su potencia, flexibilidad y seguridad. Estos son resultado de numerosas características integradas al sistema, las que están disponibles tan pronto como se inicia: multitarea, multiusuario, shells programables, independencia de dispositivos, comunicaciones y capacidades de red, y portabilidad de sistemas abiertos.

Si estamos presentes frente a la problemática de que la estrategia es mantener una línea de sistemas operativos de altas prestaciones (Unix, Linux) con modestos recursos informáticos para la capa de servidores y que actualmente el principal desarrollo de software se realiza utilizando la tecnología .NET de Microsoft; provocando que las aplicaciones creadas por los equipos de desarrollo de software en la actualidad sean incompatibles con el CLI de Mono, impidiendo esto la portabilidad de las mismas hacia plataforma Linux. Entonces, nuestro **Problema Científico** sería: ¿Cómo lograr la portabilidad, compatibilidad y despliegue en plataformas Linux a través del CLI desarrollado por el proyecto Mono de productos de software creados con la tecnología .NET de Microsoft?

Objeto de estudio

Herramientas y alternativas utilizadas para lograr la portabilidad sobre Linux de sistemas Web desarrollados con la tecnología .NET de Microsoft.

Campo de acción

Herramientas y alternativas desarrolladas por el proyecto Mono para lograr la portabilidad sobre Linux de sistemas Web desarrollados con la tecnología .NET de Microsoft.

Objetivo general

Proponer una estrategia de portabilidad que permita desplegar aplicaciones Web soportadas sobre tecnología .NET de Microsoft en plataformas Linux, utilizando Mono.

Objetivos específicos

- Hacer un estudio del estado del arte relacionado con el proyecto Mono.
- Exponer las limitaciones actuales.
- Definir los requerimientos de portabilidad que deben cumplir las aplicaciones desarrolladas con .NET de Microsoft para lograr que sean portables hacia Linux.
- Presentar las perspectivas de desarrollo futuro.

Idea a defender

Contar con una estrategia de portabilidad a seguir durante el desarrollo de aplicaciones Web con la tecnología .NET de Microsoft ayudará a garantizar el despliegue de las mismas en Linux.

Tareas investigativas

1. Estudiar y asimilar la arquitectura tecnológica que se investiga.
2. Implementar y configurar el soporte (Hardware y Software) para las pruebas de la infraestructura tecnológica.
3. Realizar pruebas de despliegue y validación.
4. Actualizar los avances, limitaciones que presenta el proyecto Mono.
5. Describir métodos y procedimientos para el despliegue de la tecnología que se investiga.
6. Describir las perspectivas futuras de la tecnología desarrollada por el proyecto Mono.

Métodos teóricos de la Investigación:

Histórico-lógico: Utilizado para estudiar la evolución y desarrollo del proyecto Mono y las diferentes herramientas creadas por este para lograr hacer compatibles con el CLI de Mono a las aplicaciones creadas con .NET de Microsoft.

Analítico-sintético: Mediante este método se procesó la información y sirvió para arribar a conclusiones en la investigación, además para determinar las actividades que debían ser incluidas dentro de la estrategia a proponer.

Métodos empíricos:

La observación: Planificada y dirigida con el fin de realizar la fundamentación teórica del problema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

Con este capítulo, se pretende mostrar elementos teóricos que son necesarios para comprender el tema tratado por la investigación, así como, un estudio del arte a nivel mundial y nacional acerca de las herramientas utilizadas para convertir portable el desarrollo de aplicaciones utilizando la tecnología .NET de Microsoft.

1.2 Plataforma

Una plataforma es una combinación de hardware y software usada para ejecutar aplicaciones; en su forma más simple, consiste únicamente de un sistema operativo, una arquitectura, o una combinación de ambos. La plataforma más conocida es probablemente Microsoft Windows en una arquitectura x86; otras plataformas conocidas son GNU/Linux y Mac OS X (que ya de por sí son multiplataforma). Hay, por otro lado, aparatos como celulares que, a pesar de ser plataformas informáticas, no se consideran usualmente como tales. El software en general está escrito de modo que dependa de las características de una plataforma particular; bien sea el hardware, sistema operativo, o máquina virtual en que se ejecuta.

1.2.1 Plataformas de hardware

Una plataforma de hardware es una arquitectura de computador o de procesador. Por ejemplo, los procesadores x86 y x86-64 son las arquitecturas más comunes actualmente para los computadores caseros. Entre los sistemas operativos existentes para estas arquitecturas se encuentran Windows, GNU/Linux, GNU/Hurd, Mac OS X, y BSD.

1.2.2 Plataformas de software

Las plataformas de software pueden ser un sistema operativo, un entorno de programación, o (más comúnmente) una combinación de ambos. Algunos ejemplos de plataformas de software son: .NET, J2EE y CORBA.

1.3 Portabilidad de un Software

La portabilidad de un software se define como, el grado de dependencia de la plataforma en la que corre. La portabilidad es mayor cuanto menor es el grado de dependencia del software de plataforma. Si un software puede ser compilado en plataformas diversas (x86, IA64, amd64, etc.), entonces se dice que el software es multiplataforma.

En algunos casos el software es "independiente" de la plataforma y puede ejecutarse en plataformas diversas sin necesidad de ser compilado específicamente para cada una de ellas, a este tipo de software se le llama interpretado, por que necesita de un intérprete para ser ejecutado en las diferentes plataformas.

1.4 ¿Qué es Microsoft .NET?

Microsoft .NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con los objetivos de:

- Mejorar sus sistemas operativos.
- Mejorar su modelo de componentes COM+.
- Obtener un entorno específicamente diseñado para el desarrollo y ejecución del software en forma de servicios que puedan ser tanto publicados como accedidos a través de la Internet de forma independiente del lenguaje de programación, modelo de objetos, sistema operativo y hardware utilizados tanto para desarrollarlos como para publicarlos. Este entorno es lo que se denomina la plataforma .NET, y los servicios antes mencionados son a los que se denomina servicios Web.

Para el desarrollo y ejecución de aplicaciones en este nuevo entorno tecnológico, Microsoft proporciona el conjunto de herramientas conocido marco de trabajo de .NET, que es posible descargarlo gratuitamente de su sitio Web e incluye compiladores de lenguajes como C#, Visual Basic.NET, C++ y JScript.NET específicamente diseñados para él.

El núcleo de la plataforma .NET lo constituye el Entorno Común de Ejecución para Lenguajes (CLR, por sus siglas en inglés), que es una aplicación similar a una máquina virtual, encarga de gestionar la ejecución de las aplicaciones escritas para ella. A estas aplicaciones les ofrece numerosos servicios que facilitan su desarrollo y mantenimiento, y favorecen su fiabilidad y seguridad. Entre estos servicios los principales son:

- Modelo de programación consistente y sencillo, completamente orientado a objetos.
- Eliminación del temido problema de compatibilidad entre DLLs conocido como "infierno de las DLLs".
- Ejecución multiplataforma.
- Ejecución multilenguaje, hasta el punto de que es posible hacer cosas como capturar en un programa escrito en C# una excepción escrita en Visual Basic.NET que a su vez hereda de un tipo de excepción escrita en Cobol.NET. Aunque más arriba se ha dicho que en el marco de trabajo de .NET sólo se ofrecen compiladores de C#, MC++, VB.NET y JScript.NET, lo cierto es que aparte Microsoft y terceros han desarrollado versiones adaptadas a .NET de muchísimos otros lenguajes como APL, CAML, Cobol, Eiffel, Fortran, Haskell, y Java, entre otros.
- Recolección de basura.
- Aislamiento de memoria entre procesos y comprobaciones automáticas de seguridad de tipos en las conversiones.
- Soporte multihilo.
- Gestión del acceso a objetos remotos que permite el desarrollo de aplicaciones distribuidas de manera transparente a la ubicación real de cada uno de los objetos utilizados en las mismas.
- Seguridad avanzada, hasta el punto de que es posible limitar los permisos de ejecución del código en función de su procedencia (Internet, red local, CD-ROM, etc.), el usuario que lo ejecuta o la empresa que lo creó.
- Interoperabilidad con código preexistente, de manera que es posible utilizar con facilidad cualquier librería de funciones u objetos COM y COM+ creados con anterioridad a la aparición de la plataforma .NET.
- Adecuación automática de la eficiencia de las aplicaciones a las características concretas de cada máquina donde se vaya a ejecutar.

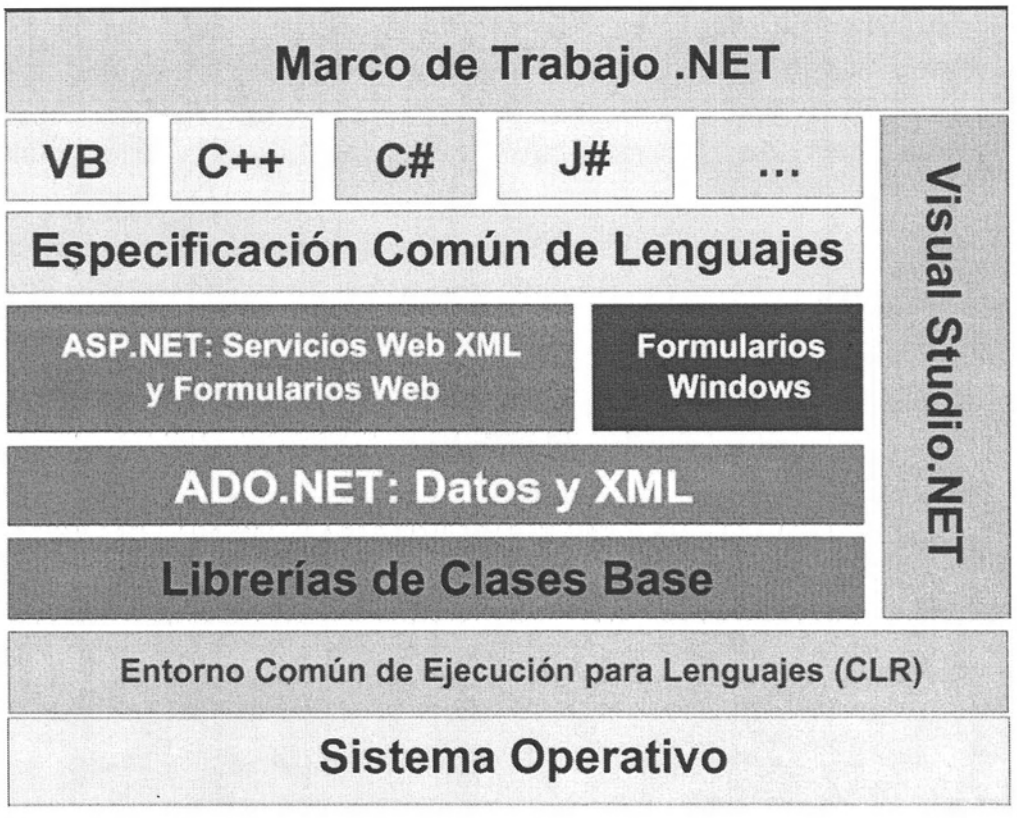

Figura 1: Marco de Trabajo .NET desarrollado por Microsoft.

1.5 Entorno de Ejecución Común para Lenguajes

El Entorno de Ejecución Común para Lenguajes (CLR, por sus siglas en inglés) constituye el verdadero núcleo del marco de trabajo de .NET, es el encargado de brindar el entorno de ejecución para correr las aplicaciones desarrolladas en .NET, ampliando además un grupo de servicios del sistema operativo.

Las herramientas de desarrollo compilan el código fuente escrito en cualquiera de los lenguajes soportados, generando un código en lenguaje intermedio, llamado Lenguaje Intermedio de Microsoft (MSIL, por sus siglas en inglés), similar a los *bytecodes* de Java. Para compilar y generar este código el compilador se basa en la Especificación Común de los Lenguajes (CLS, por sus siglas en inglés), la cual determina las reglas necesarias para que el código MSIL sea compatible con el CLR.

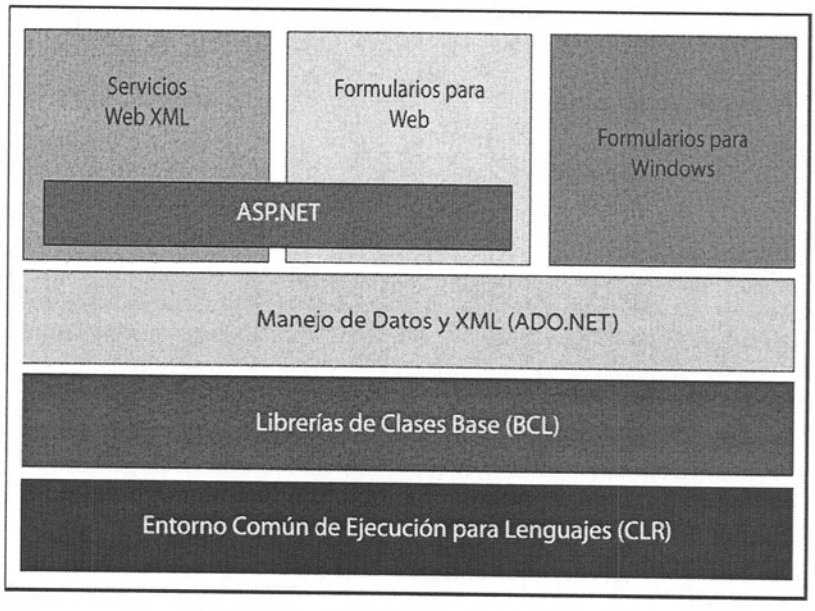

Figura 2: Diagrama básico de la biblioteca de clases base.

Esencialmente, el CLR es una máquina que es responsable de ciertos servicios de ejecución, como son: manejar y ejecutar código, implementar seguridad, administrar la memoria, entre otros servicios.

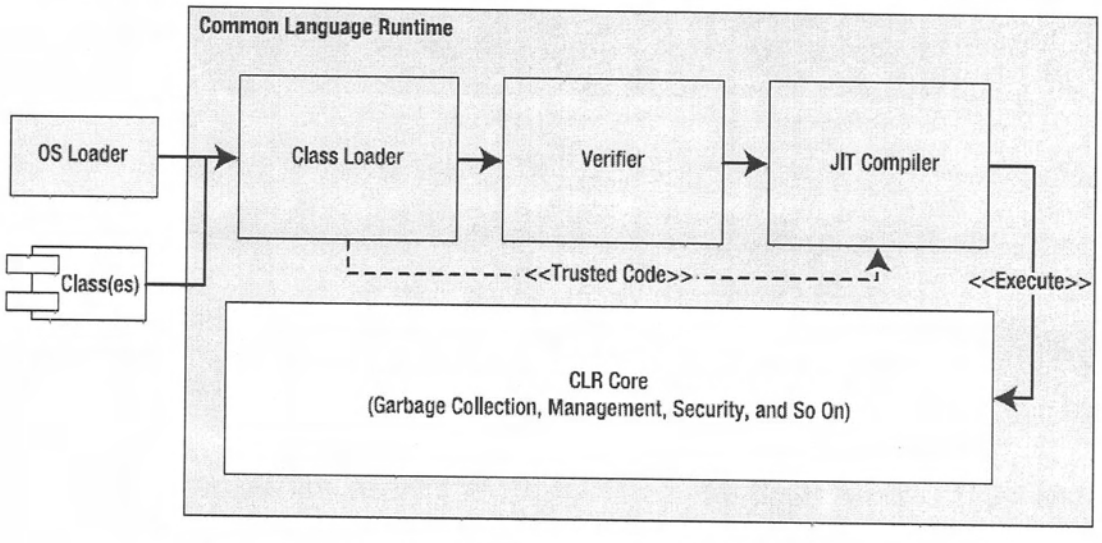

Figura 3: Entorno de Ejecución Común de los Lenguajes.

Los componentes claves dentro del CLR son:

- Formato de archivos Ejecutables Portables.
- Metadatos.
- Lenguaje Intermedio.
- El Cargador de Clases.
- El Verificador.
- El compilador JIT.
- El núcleo del CLR.

1.5.1 Formato de archivos Ejecutables Portables

Un ejecutable de Windows, EXE o DLL, tiene que estar conformado por un formato de fichero llamado, formato de archivo Ejecutable Portable (PE, por sus siglas en inglés), el cual se deriva del formato para ficheros ejecutables conocido como, Formato Común para Objetos de Ficheros (COFF, por sus siglas en inglés). Ambos formatos están completamente especificados y públicamente disponibles. El sistema operativo Windows sabe como cargar y ejecutar DLLs y EXEs porque conoce el formato de los ficheros PE. Como resultado, cualquier compilador que quiera generar ejecutables de Windows tiene que obedecer la especificación PE/COFF, que rige su formato.

Un archivo PE estándar esta dividido en un número de secciones, comenzando por un encabezado (header) en MS-DOS, luego le sigue un encabezado PE, seguido por un encabezado opcional, y luego vienen un número de secciones de imagen nativas, incluyendo las secciones: **.text, .data, .rdata** y **.rsrc**. Estas son las secciones típicas de un ejecutable de Windows.

Para soportar el CLR, Microsoft ha extendido el formato de fichero PE/COFF incluyéndole metadatos y código en Lenguaje Intermedio (IL, por sus siglas en Inglés), debido a que el CLR utiliza los metadatos para determinar cómo cargar las clases y utiliza el código IL convirtiéndolo a código nativo para la ejecución de los programas.

Por tanto, los ejecutables construidos por la plataforma .NET son un poco diferentes a los típicos ejecutables de Windows, debido a que no sólo comprenden código y datos, sino también metadatos, o sea, información sobre los datos. Estos ejecutables están dispuestos a través de ensamblados

(assemblies), los cuales no son más que unidades básicas de despliegue y versionado, consistiendo de un manifiesto, uno o más módulos y un grupo opcional de recursos.

1.5.2 Metadatos

Para que dos componentes, sistemas u objetos puedan interactuar entre sí, al menos uno debe conocer al otro. En COM, esto era permitido realizando especificaciones de las interfaces, las cuales contenían los prototipos de los métodos, incluyendo las definiciones para todos los parámetros y los tipos de retorno. Ahora, debido a que el lenguaje para definir estas especificaciones, conocido como Lenguaje de Especificación de Interfaces (IDL, por sus siglas en inglés), era demasiado complejo, Microsoft tuvo que darle una solución a esto, creando lo que ellos llamaron "librerías de tipo". Las librerías de tipo permitían a un ambiente de desarrollo o herramienta leer, realizar ingeniería inversa y crear envolturas de clases más apropiadas para el desarrollador en un lenguaje específico. Las librerías de tipo se convirtieron en una solución para COM muy buena, pero muchos desarrolladores comenzaron a quejarse de su falta de estandarización, fue así que, el equipo de .NET tuvo que inventar un nuevo mecanismo para capturar información de tipos; a este nuevo mecanismo lo llamaron metadatos.

Los metadatos no son más que, información sobre un recurso que puede ser leída por la máquina, o datos acerca de los datos. Esta información contendrá detalles sobre el contenido, tamaño, formato y otras características más del recurso (1).

Los metadatos proporcionan la información suficiente para cualquier entorno de ejecución (runtime), necesaria para localizar literalmente todo lo que este necesita para integrar los componentes. Por ejemplo:

- El CLR utiliza metadatos para la verificación, la seguridad y la localización de recursos del contexto.
- El cargador de clases utiliza metadatos para buscar y cargar las clases .NET.
- El JIT los usa para compilar el código IL.

Algunas herramientas usan metadatos para soportar integración. Por ejemplo, algunas herramientas los utilizan para hacer llamadas a envolturas creadas con .NET y componentes COM, permitiendo así su interoperabilidad.

1.5.3 Lenguaje Intermedio

En la ingeniería de software, el concepto abstracción es extremadamente importante. Usualmente utilizamos la abstracción para ocultar la complejidad de los sistemas y servicios de las aplicaciones, proporcionando en cambio una interfaz simple al consumidor. De esta forma podemos realizar cambios en la implementación interna, sin tener que cambiar la interfaz, así los consumidores siempre podrán usar la misma interfaz sin necesidad de preocuparse por los cambios internos de los servicios.

En lenguajes más avanzados, los desarrolladores comenzaron a preocuparse por introducir una capa de abstracción de lenguaje, ejemplo: *p-code* y *bytecode*. Producido por el compilador de Pascal, *p-code* es un lenguaje intermedio que soporta programación procedural; igualmente, generado por los compiladores de Java, *bytecode* es un lenguaje intermedio que soporta programación orientada a objetos. *Bytecode* es una abstracción de lenguaje que permite que el código de Java sea capaz de correr en diferentes plataformas, mientras estas plataformas tengan una máquina virtual que se encargue de ejecutar dichos *bytecodes*.

Microsoft llama a su propia capa de abstracción de lenguaje, Lenguaje Intermedio de Microsoft (MSIL, por sus siglas en inglés) o simplemente de una forma más abreviada, IL. IL nos es más que una implementación del Lenguaje Intermedio Común (CIL: de sus siglas en inglés), un elemento clave de la especificación ECMA 335 del CLI. Similar a los *bytecodes*, IL soporta todas las características orientadas a objetos, incluyendo abstracción de datos, herencia, polimorfismo, y conceptos útiles como el manejo de excepciones y eventos. Además de estas características, soporta otros conceptos, tales como: propiedades, campos y enumeraciones. Cualquier lenguaje .NET debe ser convertido a IL, por consiguiente esto permite que .NET soporte múltiples lenguajes, así como múltiples plataformas, mientras estas tengan presente un CLR que se encargue de la ejecución.

1.5.4 El Cargador de Clases

Cuando usted corre una aplicación estándar de Windows, el cargador del sistema operativo es el responsable de cargar la aplicación antes de que sea ejecutada. Este proceso se realiza de la siguiente manera; al correr una aplicación .NET, el cargador del sistema operativo reconoce que es una aplicación .NET y entonces pasa el control de dicha aplicación al CLR. El CLR encuentra el punto de entrada para la ejecución de la aplicación, el cual es típicamente el `Main()`, y lo ejecuta para iniciar la aplicación. Pero antes de que el `Main()` pueda ejecutarse, el cargador de clases (class loader)

tiene que encontrar la clase que implementa al método `Main()` y cargarla. Además, cuando el `Main()` instancia un objeto de una clase específica, el cargador de clases también es el responsable de localizarla y cargarla. El cargador de clases desarrolla este proceso cada vez que un tipo es referenciado por primera vez.

Por lo tanto, el cargador de clases es el encargado de cargar las clases en memoria y prepararlas para la ejecución. Antes de que haga esto exitosamente, debe localizar la clase específica. Para poder encontrar la clase, este tiene que realizar una búsqueda en diferentes lugares, incluyendo los ficheros de configuración de la aplicación (.config) localizados en el directorio actual, la Caché Global para Ensamblados (GAC, por sus siglas en inglés), y el metadato que contiene el fichero PE, específicamente el manifiesto.

Una vez que el cargador de clases ha encontrado y cargado la clase especificada, este cachea la información de tipo de la clase, de manera tal que no sea necesario cargarla otra vez durante la ejecución del mismo proceso de la aplicación. Teniendo esta información, más tarde podrá determinar cuanta memoria será necesario reservar para crear una nueva instancia de la clase. Después de haber cargado la clase especificada, el cargador de clases inyecta una pequeña rutina (stub), como una función Prolog, dentro de cada método de la clase cargada. Esta rutina es usada con dos propósitos: denotar el estado de la compilación JIT y para la transición entre código manejado y código no manejado. Si la clase cargada referencia otras clases, el cargador tratará también de cargar los tipos referenciados. Sin embargo, si los tipos referenciados han sido cargados anteriormente, el cargador de clases no tendrá nada que hacer. Finalmente, el cargador de clases usa los metadatos apropiados para inicializar las variables estáticas e instanciar un objeto de la clase cargada.

1.5.5 El Verificador

Los lenguajes script e interpretados son bastante flexibles con el uso de tipos, permitiendo escribir código sin tener que declarar explícitamente las variables. Esta flexibilidad puede introducir código extremadamente expuesto a errores y difícil de mantener, y esto provoca que misteriosamente el programa pueda fallar durante la ejecución. A diferencia de los lenguajes script e interpretados, los lenguajes compilados requieren que los tipos tengan que ser explícitamente definidos a priori para su uso, permitiendo al compilador asegurar que los tipos están usados correctamente y que el código va a poder ser ejecutado sin riesgos de errores debido al uso de los tipos de datos.

La clave es la seguridad de tipo, y constituye un concepto fundamental para la verificación de tipo en .NET. Dentro de la Máquina Virtual de Ejecución (VES, por sus siglas en inglés), el verificador es el componente que se ejecuta en tiempo de corrida para verificar que el código es tipo seguro (type safe). Notemos que esta verificación de tipo es realizada en tiempo de corrida, y que esto es una diferencia fundamental entre .NET y otros entornos de ejecución. Verificando la seguridad de tipo en tiempo de corrida, puede prevenir la ejecución de código que no es tipo seguro, y asegurar que el código será ejecutado correctamente. En breves palabras, seguridad de tipo significa más confiabilidad.

Ahora entenderá como el verificador encaja dentro del CLR. Después que el cargador de clases ha cargado una clase y antes de que cualquier pedazo de código IL pueda ser ejecutado, el verificador chequea todo el código que tiene que ser verificado. El es el responsable por verificar que:

- Los metadatos son bien formados, o sea, que los metadatos son válidos.
- El código IL es tipo seguro, queriendo decir que la firma de los tipos está siendo usada correctamente.

Estos dos criterios tienen que ser conocidos antes de que el código pueda ser ejecutado, ya que, la compilación JIT solo se realizará cuando el código y los metadatos hayan sido verificados satisfactoriamente. Además de chequear la seguridad de tipos, el verificador también desarrolla un análisis rudimentario del control de flujo del código para asegurarse que el código está usando los tipos correctamente. Aunque el verificador forma parte del compilador JIT, solamente realiza su función cuando un método es invocado, no cuando una clase o un ensamblado es cargado. La verificación constituye un paso opcional porque el código seguro nunca va a ser verificado, sino que será inmediatamente dirigido al JIT para que realice su compilación.

1.5.6 El compilador JIT

Los compiladores JIT (Just-In-Time) juegan un mayor papel en la plataforma .NET debido a que todos los archivos PE de .NET, como se hizo referencia anteriormente, contienen código IL y metadatos, no código nativo. Los compiladores JIT son los encargados de convertir el código IL en código nativo, permitiendo que este código pueda ser ejecutado en un sistema operativo en específico. Cada método que ha sido verificado por la seguridad de tipo, es compilado por el compilador JIT del CLR y luego convertido en código nativo.

Una ventaja que comprende el compilador JIT es que puede compilar dinámicamente el código, esto es óptimo para la máquina en específico. Si colocamos un fichero PE de .NET de una primera máquina en una segunda máquina, el compilador JIT en la segunda máquina conoce las características de su CPU, y será capaz de separar el código nativo mas optimizado para su CPU. Otra ventaja considerable es que se puede tomar el mismo fichero PE y correrlo en plataformas totalmente diferentes, mientras esa plataforma tenga un CLR.

Por razones de optimización, la compilación JIT ocurre solamente la primera vez que un método es invocado. Retomando que el cargador de clases añade una rutina a cada método durante la carga de la clase. Cuando ocurre la primera invocación del método, la Máquina Virtual de Ejecución lee la información contenida en la rutina, la cual indica que el código del método no ha sido compilado por el JIT todavía, con esta indicación, el compilador JIT compila el método e inyecta la dirección del método nativo dentro de la rutina. Durante subsecuentes invocaciones al mismo método, no hará falta una compilación JIT de nuevo porque cada vez que la Máquina Virtual de Ejecución verifique la rutina, esta encontrará la dirección del método nativo. Como el compilador JIT desarrolla esta tarea la primera vez que un método es invocado, los métodos que no se necesitan en tiempo de corrida nunca serán compilados por el JIT.

El compilado o código nativo permanece en memoria hasta que el proceso es terminado y el recolector de basura limpia todas las referencias y la memoria asociada con el proceso. Esto significa que la próxima vez que se ejecute el proceso o el componente, el compilador JIT volverá a desarrollar todas sus tareas nuevamente.

1.5.7 Soporte y administración de la ejecución

Hasta el momento pueden percibir que cada componente del CLR visto usa de alguna forma metadatos y código IL para poder llevar a cabo satisfactoriamente los servicios que este soporta. Además de proveer metadatos y el código manejado generado, el compilador JIT tiene que generar datos manejados (managed data) que el administrador de código (code manager) necesita para localizar y administrar los marcos de memoria de la pila (stack). El administrador de código utiliza datos manejados para controlar la ejecución del código, incluyendo búsquedas en la pila que son requeridas para el manejo de excepciones, chequeo de seguridad, y recolección de basura. Además del administrador de código, el CLR también brinda un número importante de soportes de ejecución y administración de servicios. A continuación enumeramos brevemente un grupo de ellos:

Recolección de basura (garbage collection)

A diferencia de C++, donde tenemos que destruir manualmente toda la memoria reservada del montón (heap), conocida también como memoria dinámica, el CLR soporta administración automática del tiempo de vida para todos los objetos de .NET. El recolector de basura puede detectar cuando un objeto no está siendo referenciado y desarrollar una tarea para liberar la memoria que no está siendo utilizada.

Manejo de excepciones (exception handling)

Antes de surgir .NET, no existía un método consistente para el manejo de errores o manejo de excepciones, causando esto un poco de problemas a la hora de manejar y reportar un error. En .NET, el CLR soporta un mecanismo estándar para el manejo de excepciones, que trabaja a través de todos los lenguajes, permitiéndole a cada programa usar un mecanismo común para el manejo de errores. Este mecanismo de manejo de excepciones está integrado al Manejo de Excepciones Estructurado de Windows (SEH: por sus siglas en inglés).

Soporte de seguridad

El CLR desarrolla varios chequeos de seguridad en tiempo de corrida para asegurar que el código es seguro y puede ser ejecutado, y que no está abriendo brecha a cualquiera de los requerimientos de seguridad. Además de soportar la seguridad de acceso al código, el motor de seguridad también soporta chequeos de seguridad declarativos y programáticos.

Soporte para la depuración y búsqueda de errores (debugging)

El CLR proporciona un rico soporte para la depuración y búsqueda de errores. Existe toda una API que los desarrolladores de compiladores pueden usar para desarrollar su propio depurador (debugger). Esta API tiene soporte para controlar la ejecución del programa, puntos de ruptura (breakpoints), excepciones, control de flujo, etc.

1.6 El Sistema de Tipos Común

.NET trata a todos los lenguajes que comprende de la misma forma, una clase escrita en C# es equivalente a una clase escrita en VB.NET, de la misma forma una definida en C++ Manejado o

COBOL Manejado. Todos estos lenguajes tienen que tener cierta correspondencia en aras de poder integrarse unos con otros. Para hacer realidad esta integración de lenguajes, Microsoft ha especificado un Sistema de Tipos Común (CTS, por sus siglas en inglés), por el cual todos los lenguajes disponibles por la plataforma deben regirse. En la siguiente imagen se pueden apreciar las dos clasificaciones que comprende el CTS.

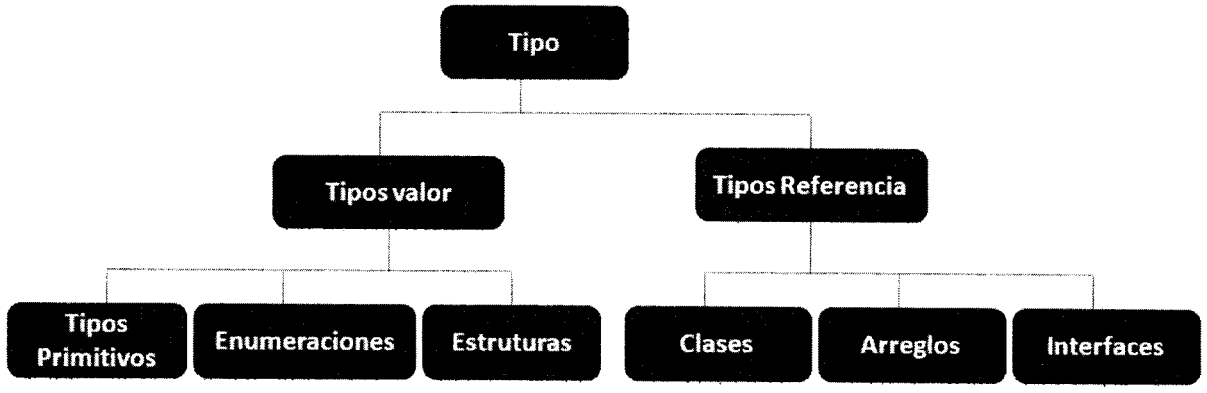

Figura 4: Clasificación de los tipos.

1.6.1 Tipos Valor

En general, el CLR soporta dos tipos diferentes: tipos valor y tipos referencia. (2) Los tipos valor representan a los valores almacenados en la pila. Estos tipos no pueden ser nulos (null) y siempre tienen que contener algún dato. Cuando un tipo valor es pasado como argumento a una función, este es pasado por valor, queriendo decir que una copia por valor del tipo será realizada antes de ejecutar la función. Esto implica que el valor original del tipo no cambia después de haber sido ejecutada la función. Como los tipos intrínsecos son pequeños en tamaño y no consumen mucha memoria, el costo de realizarle una copia es despreciable. Dentro de los tipos valor se encuentran: los tipos primitivos o intrínsecos, las estructuras y las enumeraciones.

También se puede crear un tipo valor derivando de la clase **System.ValueType**. Algo importante que debe conocer es que los tipos valor son sellados (sealed), esto quiere decir que cuando una clase derive de **System.ValueType**, ninguna otra clase podrá derivar de la clase creada.

1.6.2 Tipos Referencia

Si un tipo consume grandes recursos de memoria, entonces en este caso un tipo referencia proporciona más beneficios que un tipo valor. Los tipos referencia se llaman así porque estos contienen referencia a objetos almacenados en el área de memoria llamada montón (heap), por lo tanto estos tipos si pueden ser nulos. Otra característica significativa de estos tipos es que son pasados a los métodos por referencia, significando que si se pasa un tipo referencia a una función, lo que estamos haciendo realmente es pasando un puntero a la dirección de memoria donde está creado el tipo, de manera que la ejecución de la función trabaja directamente con el objeto original pasado como argumento.

El primer beneficio de los tipos referencia es que pueden ser utilizados en una función como parámetros de salida, y el segundo beneficio es que no se gastan recursos extras, ya que no se realiza una nueva copia del tipo cada vez que se pasa como argumento a una función. Si un objeto es considerablemente grande (consume mucha memoria), entonces la mejor opción es que sea un tipo referencia. En .NET, una de las desventajas de los tipos referencia es que estos tienen que ser almacenados en el montón manejado, lo que significa que requieren más ciclos de CPU porque tienen que ser administrados y recolectados por el CLR. Entre los tipos referencia se encuentran: las clases, los interfaces y los arreglos.

1.7 Especificación Común de Lenguajes

Una de las metas de .NET es lograr la integración de lenguajes, de tal forma que los programas puedan ser escritos en cualquier lenguaje e interoperar un lenguaje con otro, aprovechando además completamente las características de la programación orientada a objetos, como son: la herencia, el polimorfismo, manejo de excepciones, etc. (2). Sin embargo, todos los lenguajes no son iguales, un lenguaje puede soportar una característica que puede ser completamente diferente en otro lenguaje.

Por ejemplo, C++ Manejado es sensible a las mayúsculas, mientras que, VB.NET no lo es. La Especificación Común de Lenguajes (CLS: por sus siglas en inglés) especifica una serie de reglas básicas que son requeridas para lograr la integración de varios lenguajes sobre la plataforma .NET. Microsoft proporciona la CLS para especificar los requerimientos mínimos que debe cumplir un lenguaje para poder ser un lenguaje .NET, de esta forma los desarrolladores de compiladores pueden construir sus propias implementaciones siguiendo esta especificación.

1.8 Estandarización de .NET

Microsoft trabajó muy duro para lograr estandarizar su CLI y su lenguaje C#; y lo lograron con resultados satisfactorios. No existe realmente un lenguaje de programación o plataforma que se haya estandarizado tan rápido desde la salida de su primera versión, como lo logró el lenguaje C# y su CLI.

1.8.1 Organizaciones de Estándares

Microsoft originalmente sometió a la estandarización el CLI de .NET y el lenguaje C#, a través de la organización internacional basada en membrecías de estándares para la comunicación y la información, *European Computer Manufacturers Association* (ECMA). En Diciembre de 2002, la segunda edición de la ECMA-335 del CLI se convirtió en un estándar. El lenguaje de programación C# se convirtió en ECMA-334 simultáneamente (1). Debido a la estrecha relación existente entre ECMA y la Organización Internacional de Estándares (ISO, por sus siglas en inglés), el CLI y C# terminaron su estandarización en Abril de 2003. El CLI está certificado bajo la norma ISO 23 271: 2003, y C# bajo la norma ISO 23 270:2003. Estas normas de estandarización ISO permiten que puedan ser creadas y usadas con gran confianza y libertad otras implementaciones del CLI y del lenguaje C#.

1.8.2 ¿Qué no está concebido dentro de los estándares?

Algunas de las partes más innovadoras del marco de trabajo de .NET de Microsoft, como son: ADO.NET, ASP.NET y Windows Forms, no están comprendidas en los estándares del CLI y de C#. Los Windows Forms son un gran avance en el rápido desarrollo y uso fácil de la programación tradicional de Interfaces Gráficas de Usuarios (GUI, según siglas en inglés). Los Windows Forms y los User Interfaces (UI) constituyen la problemática mayor para poder convertir a .NET independiente de plataforma (1).

Por otra parte se encuentra ADO.NET, otra de las tecnologías no estandarizadas. ADO.NET mezcla XML y acceso a datos muy bien, permitiendo que sistemas multicapas puedan ser implementados de forma más fácil. Desafortunadamente, la no estandarización de ADO.NET realmente impide el desarrollo puro de Servicios Web (Web Services), ya que los datasets no pueden ser transferidos de un Servicio Web implementado en .NET a otro implementado en otra plataforma, ejemplo Java. Por último, la otra tecnología que tampoco forma parte de las estandarizaciones CLI y C#, es ASP.NET. La idea principal de ASP.NET es utilizar en páginas Web controles del lado del servidor, los cuales hacen

que el modelo de programación Web sea semejante al utilizado para construir aplicaciones escritorio (desktop). Esto permite una codificación más rápida de las UI Web, enriqueciendo de esta forma las UI de los clientes.

Además de las cosas que no aparecen en los estándares, hay algunas que aunque aparecen, no están definidas muy claro. Un ejemplo de ello lo forman los Platform Invoque (P/Invoque). Cuando Microsoft creó .NET sabía que no podía ser capaz de remplazar completamente todas las funcionalidades existentes pertenecientes a otros lenguajes y que también los desarrolladores iban a querer usar código legado desde .NET. Por lo tanto, para permitir todo lo expuesto anteriormente, Microsoft creó los llamados P/Invoque. Los P/Invoque permiten realizar llamadas a librerías existentes o al Sistema, para utilizar funcionalidades adicionales que .NET no tiene implementadas, ejemplo, la comunicación a través de un puerto serie. Esta característica es uno de los problemas que dificultan la portabilidad de las aplicaciones .NET hacia otras plataformas, como: Linux, Unix, Mac OS, etc.

1.9 Implementaciones del CLR de .NET más conocidas

Existen varias implementaciones del CLR de .NET, todas implementan de una forma u otra la mayor parte de las características estandarizadas bajo la norma ECMA-335 y ECMA-334 vistas anteriormente. A continuación, conocerá cuales son las principales características que comprenden estas implementaciones.

1.9.1 CLR de .NET implementado por Microsoft

La implementación generalmente disponible y más amplia de .NET es sin dudas la desarrollada por Microsoft. Por supuesto, tiene sentido que los propios creadores del estándar tengan también la primera implementación. Además del lenguaje C#, Microsoft proporciona C++.NET, Jscript.NET, y J#. .NET trabaja desde la versión de Windows 98 hasta la versión Windows Vista en la actualidad. Siendo una implementación bastante grande, .NET comprende más funcionalidades implementadas que cualquier otro CLI disponible actualmente.

La Plataforma de Desarrollo de Software de .NET (SDK, por sus siglas en inglés) se encuentra disponible libremente desde el sitio de Microsoft y trae incorporados además muy buenas herramientas para los desarrolladores y uno de los mejores depuradores gráficos actualmente disponibles.

1.9.2 Compact Framework

Un año después de ser liberado el Framework de .NET, Microsoft liberó su implementación Compact Framework (CF), la cual es una implementación del CLI específica para dispositivos Windows CE. Aunque esta implementación solo cubre un pequeño grupo de las características especificadas por el CLI, el Compact Framework permite programar para equipos móviles muy fácilmente, y están disponibles como lenguajes, C# y VB.NET. Aunque, una problemática existente es que cada dispositivo específico con procesador diferente tiene que ser compilado para ese dispositivo en específico. Visual Studio .NET convierte esta programación un poco más fácil utilizando emuladores para los dispositivos y un depurador integrado en los dispositivos a través de ActiveSync. Probablemente la característica más significativa es la posibilidad de consumir Servicios Web en un Ayudante Personal Digital (PDA, por sus siglas en inglés). También permite la integración de estos dispositivos con SQL Server CE.

Por otro lado, la implementación CF brinda poco soporte de XML, no tiene Remoting, Printing y tampoco comprende un grupo más de características disponibles en el CLR de .NET.

1.9.3 SSCLI/Rotor

Microsoft en colaboración con Corel creó la implementación Shared Source CLI (SSCLI), también conocida como Rotor. El SSCLI basa su implementación cumpliendo muy bien con los estándares (excepto en la implementación de Jscript) y fue construido para el uso y estudio académico acerca de cómo implementar un CLI independiente de plataforma. Aunque útil para el estudio, hay que decir, que el Recolector de Basura de la implementación Rotor no es recomendado para el desarrollo de aplicaciones comerciales (3). Rotor puede correr bajo las plataformas FreeBSD, Windows y OS X.

La funcionalidad de bajo nivel de la implementación Rotor está contenida en la *Platform Adaptation Layer* (PAL). La PAL esta implementada mayormente en C++ y C. Sin embargo, las librerías de entorno de ejecución de Rotor están implementadas en C#, lo cual permite profundizar en el conocimiento y uso del lenguaje C#. La estructura anterior simplifica el trabajo requerido para hacer portable a Rotor de un sistema operativo a otro, simplemente teniendo que realizar mínimos cambios en la implementación de la PAL. Como se hizo alusión anteriormente, SSCLI va más allá de la implementación del estándar CLI, ya que, implementa también Jscript, además de soportar UIs multiplataforma usando el toolkit TCL/TK.

1.9.4 Portable.NET

Portable.NET es una implementación código abierto del CLI, promovida por el proyecto GNU y el comité dotGNU. Esta implementación se enfoca principalmente al igual que SSCLI en los estándares del CLI, pero además implementa funcionalidades adicionales usando TCL/TK para el desarrollo de UIs y usando algunas implementaciones de Mono (1). Actualmente, Portable.NET corre en plataformas Linux, Windows, Solaris y Mac OS X. Sus librerías de bajo nivel, así como, su entorno de ejecución y compilador están escritas en el lenguaje de programación C. Una de las características más significativas y de mayor contraste de la implementación Portable.NET es que lo primero que construyeron fue un intérprete. Pero, no todas sus implementaciones tienen un intérprete, sino un compilador JIT. Las implementaciones que usan un compilador JIT son considerablemente rápidas, además son las más pequeñas y portables. Portable.NET está construido bajo la licencia código abierto GNU GPL.

1.9.5 Mono

Mono es un proyecto de implementación del marco de trabajo de .NET desarrollado por Microsoft utilizando código libre, gestionado por Ximian y basado en las especificaciones definidas en ECMA. Los objetivos en un inicio del proyecto Mono fueron, implementar en un entorno de software libre para el mundo Unix la especificaciones ECMA, para lo cual se incluye un compilador para C#, un entorno de ejecución CLR y un conjunto de librerías de clases que incluyen las Librerías de Clases del marco de trabajo (FCL, por sus siglas en Inglés), así como otras añadidas.

Ximian (actualmente Novell) no tenía el tamaño ni los recursos suficientes como para abordar un proyecto de esta envergadura por si sola; por lo tanto, cuando el proyecto alcanzó cierto nivel de realización fue abierto a la comunidad para buscar nuevos colaboradores. El interés que suscitó el proyecto desde un comienzo fue muy grande, lo que hizo que multitud de desarrolladores se involucraran en el proyecto. Este interés provocó también que partes de la tecnología, como ADO.NET, ASP.NET o WinForms, que inicialmente no estaban contempladas, o no eran prioritarias, se comenzasen a implementar.

Licencias

Mono usa tres tipos de licencia. Las librerías de clases están bajo una licencia X11. Este tipo de licencia permite prácticamente cualquier uso del código, incluido copiarlo y usarlo en una aplicación propia. El único requisito es que se mantenga la información de copyright de los archivos. No es necesario siquiera indicar que se está utilizando software bajo licencia X11.

Las librerías del entorno de ejecución, como el JIT, se distribuyen bajo licencia LGPL. De esta forma si se realiza un programa que enlace con ellas, se puede mantener bajo código propietario. Sin embargo, es necesario permitir que se pueda enlazar con versiones más recientes de las librerías. La forma más fácil de hacer esto sería enlazando dinámicamente con ellas. Esto obligaría al usuario a descargar e instalar mono por separado. El resto de aplicaciones, como `mono` o `mcs`, usan una licencia GPL. Por lo tanto cualquier aplicación que esté basada en ella, deberá mantener esta licencia.

En lo que respecta al cumplimiento de las patentes de software, las partes contempladas en el estándar ECMA no tienen ningún problema, ya que, se permite a cualquiera implementar esos componentes gratuitamente y para cualquier propósito. Sin embargo, cuestiones referentes a ADO.NET, ASP.NET y WinForms, son bastante diferentes. La estrategia de Mono ante estas tecnologías es la siguiente:

- Evitar la patente utilizando otros mecanismos para implementar la misma funcionalidad.
- Eliminar las porciones de código bajo patente.
- Encontrar algo más novedoso que deje sin uso a la patente.

A continuación se muestra un gráfico que representa algunos de los diferentes tipos de tecnologías que se están implementando en Mono. Los elementos, tales como: la máquina del entorno de ejecución, las clases del núcleo y las clases para el manejo de Xml; son los correspondientes a ECMA y que no están bajo ningún tipo de patente.

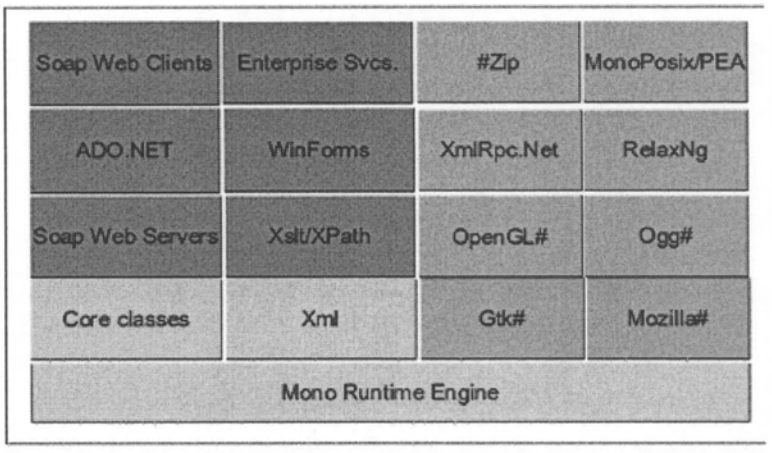

Figura 5: Principales tecnologías implementadas por Mono.

1.10 ¿Qué IDEs están disponibles para desarrollar con Mono?

Actualmente, entre los IDEs más utilizados para desarrollar con Mono existen:

- MonoDevelop
- Eclipse
- X-Develop
- SharpDevelop

1.10.1 MonoDevelop

MonoDevelop está basado en un IDE código abierto desarrollado previamente llamado SharpDevelop (también conocido como #develop). Este era un entorno de desarrollo para C# y VB.NET disponible para ser descargado gratuitamente. Su código fuente fue tomado y exportado usando las librerías del marco de trabajo de GNOME (Gtk+), con el objetivo de que usara la plataforma de desarrollo desarrollada por Mono para la compilación (2). De esta forma surge MonoDevelop, el cual se encuentra actualmente en la versión 1.0.

MonoDevelop integra características similares a las que comprenden otros IDEs (VS .NET, Eclipse, etc.), como son: intellisense, control e integración de código fuente, GUI integrados y un diseñador Web. Tiene soporte para los lenguajes de programación C#, VB.NET, Java, Boo, Memerle y C++.

1.10.2 Eclipse

Eclipse es un IDE de software de código abierto independiente de plataforma, para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar IDEs, como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador *Embedded Java Controller* (ECJ), que se entrega como parte de Eclipse; y que son usados también para desarrollar el mismo Eclipse. Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus.

Eclipse comprende también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Project, cubriendo casi todas las áreas de la Ingenieria Dirigida por Modelos (MDE, por sus siglas en inglés). Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. El desarrollo de este IDE se encuentra en la versión 3.2.

1.10.3 X-Develop

X-Develop es un IDE comercial, multiplataforma y multilenguaje desarrollado por la compañía Omnicore. Este proporciona muy buenas características para la escritura, búsqueda y mantenimiento de código. Puede ser utilizado para trabajar en la plataforma .NET, Mono o Java en Windows, Linux y en Mac OS X. Los lenguajes que comprende son: C#, Java, JSP, J#, VB.NET, JavaScript, XML y HTML (4). Se dice que este IDE es una mezcla entre Eclipse y Visual Studio, conservando lo mejor de cada uno de ellos, aunque aún no supera a ninguno de los dos. Actualmente se encuentra en la versión 2.0.

1.10.4 SharpDevelop

SharpDevelop o #develop es un IDE libre y código abierto para los lenguajes de programación C#, VB.NET y Boo. Es típicamente utilizado como una alternativa del VS .NET de Microsoft. Desde sus inicios su desarrollo tuvo una bifurcación con el desarrollo de Mono/GTK# y MonoDevelop para incluir soporte multiplataforma. Para su completamiento de código, SharpDevelop utiliza su propio parser de

C# y VB.NET, los cuales fueron generados siguiendo las versiones modificadas y descripciones de gramática del generador de compiladores Coco/R desarrollado por la universidad de Linz. Su código fuente contiene este generador.

Para el desarrollo con Boo utiliza el propio parser del compilador de Boo, pero la resolución de expresiones de tipo y la inferencia de tipos es realizada con código personalizado, que soporta evaluación perezosa de tipos. Actualmente, SharpDevelop se encuentra en la versión 3.0.

1.11 Servidores Web para ASP.NET

Primeramente debe conocer que es un servidor Web. Un servidor Web es un programa que implementa el protocolo HTTP *(Hypertext Transfer Protocol)*. Este protocolo está diseñado para transferir lo que llamamos hipertextos, páginas Web o páginas HTML *(Hypertext Markup Language)*: textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de música. Sin embargo, el hecho de que HTTP y HTML estén íntimamente ligados no debe dar lugar a confundir ambos términos. HTML es un lenguaje de marcas y HTTP es un protocolo.

Un servidor Web se encarga de mantenerse a la espera de peticiones HTTP llevadas a cabo por un cliente HTTP que solemos conocer como navegador. El navegador realiza una petición al servidor y este le responde con el contenido que el cliente solicita. Para hospedar páginas ASP.NET existen varios servidores Web, entre los más utilizados se encuentran:

1.11.1 Internet Information Server o Services (IIS)

IIS es el servidor implementado por Microsoft y el más utilizado para hospedar páginas ASP.NET, debido a que fue implementado para este fin principalmente. Internet Information Services (IIS) 6.0 es un potente servidor Web que ofrece una infraestructura de gran fiabilidad, capacidad de manejo y escalabilidad para aplicaciones Web sobre todas las versiones de Windows Server 2003.

IIS hace posible que las organizaciones aumenten la disponibilidad de sus sitios y aplicaciones Web y a la vez reducir sus costes administrativos. IIS 6.0 soporta la Iniciativa de Sistemas Dinámicos de Microsoft (DSI, por sus siglas en inglés) con monitorización de estado de salud automático, aislamiento de procesos y capacidades de gestión mejoradas (5).

1.11.2 Cassini

Cassini, es un servidor Web de ejemplo desarrollado utilizando el marco de trabajo de .NET de Microsoft. Cassini permite publicar páginas ASP.NET haciendo uso de las APIs de ASP.NET (System.Web.Hosting) (6). Fue desarrollado bajo la licencia *Common Shared Source* y fue creado por Microsoft como un ejemplo para demostrar las cosas que podían lograrse con el marco de trabajo .NET.

En general es un servidor Web completo, soporta la mayoría de los tipos MIME, HTTP 1.1 y ASP.NET. Lo más importante, su tamaño de descarga equivale a los 50KB, constituido por 10 archivos fuente y 7 clases principales. Uno de los problemas que presenta el Cassini original es que fue diseñado para escuchar solo desde localhost. Carece de una interface de administración como la de IIS, además de tener un par de problemas más de escalabilidad. Es por eso que otros desarrolladores se encargaron de tomar el código de Cassini y crear un par de versiones especializadas, es así como nacen CassiniEx y UltiDev Cassini. Este servidor fue la base para el servidor Web personal que trae Visual Studio 2005. Es importante que conozca que Cassini y sus versiones especializadas no corren en Mono (7).

Por último, el mayor valor que tiene Cassini (más de servir como un servidor Web para máquinas virtuales internas), es el educativo. Se puede aprender muchísimo del código fuente de Cassini, inclusive hay tutoriales sobre extensiones hechas para el servidor en sí.

1.11.3 Apache

Apache es actualmente el servidor Web más utilizado, de acuerdo a una entrevista realizada por la compañía Netcraft. Esta compañía muestra periódicamente estadísticas de los servidores Web más utilizados en la actualidad. El siguiente gráfico muestra estadísticas publicadas correspondientes desde el mes de Agosto de 1995, hasta al mes de Marzo del 2008.

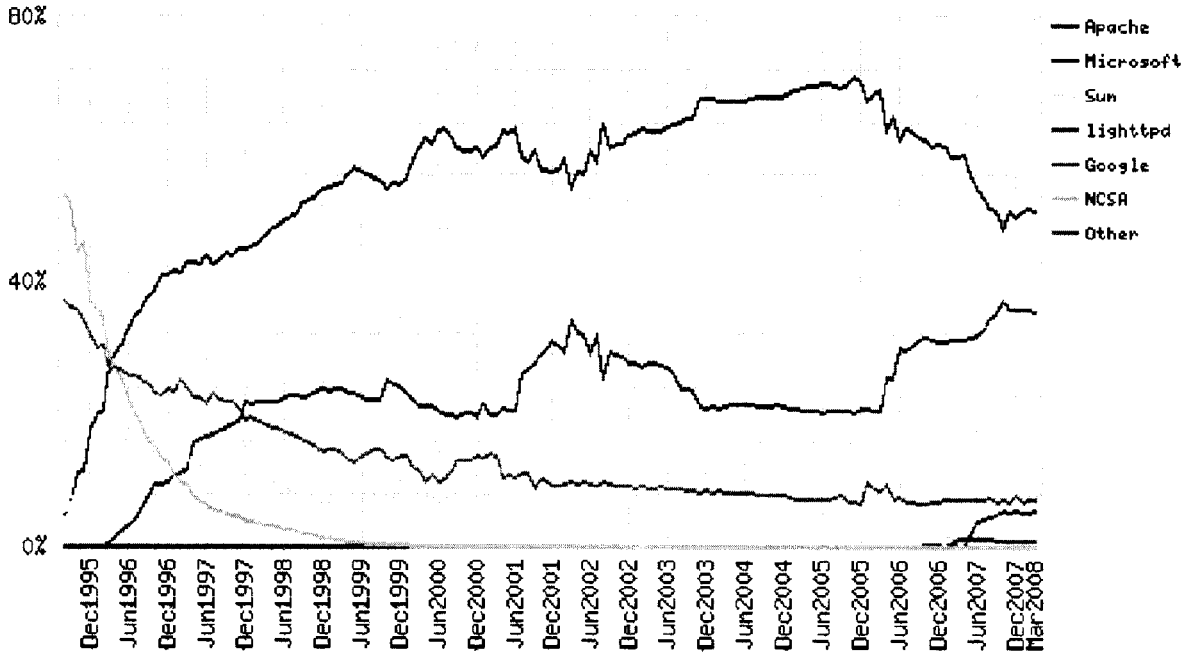

Figura 6: Servidores Web más utilizados (8).

En los primeros días de la Red, NCSA *(National Center for Super Computing Applications)* creó un servidor Web que se convirtió en el número uno de los servidores Web a principios de 1995. Los usuarios de dicho servidor comenzaron a intercambiar los parches que tenían y en breve se dieron cuenta de que había que crear un foro que se encargase de la administración de dichos parches. Había nacido Apache Group. Utilizaron el código del servidor Web de NCSA y crearon un nuevo servidor al que llamaron Apache. La primera versión (0.6.2) apareció en Abril de 1995. La versión 1.0 el 1ero de Diciembre de ese mismo año (9). Actualmente existe la versión 2.2.

Ventajas al utilizar el servidor Web Apache

A continuación se muestran algunos de los aspectos que hacen de Apache un servidor Web tan popular y a su vez el más utilizado en el mundo.

- **Apache es un servidor altamente configurable de diseño modular:** Las prestaciones de Apache son muy sencillas de ampliar en dependencia de las necesidades de quien lo utilice. Sus creadores aseguran que con un nivel aceptable en los lenguajes de programación C o Perl es posible desarrollar un módulo para que realice una función deseada. Esto trae consigo que son muchos los módulos para Apache que han sido desarrollados a lo largo de su historia.

- **Apache es una tecnología gratuita, de código fuente abierto:** Apache va más allá de ser un software gratuito, su código fuente es abierto, este código fuente se puede usar, modificar y distribuir de forma gratuita. Todo lo relacionado con las licencias de Apache se puede ampliar en la siguiente dirección (*http://www.apache.org/licenses/*).
- **Apache funciona en Linux, en otros sistemas de Unix y Windows:** A pesar de que muchos autores catalogan a Apache como uno de los mayores logros del mundo del Software Libre, este no sólo corre en sistemas operativos de esta línea. Apache aunque se considera que es más estable y eficiente cuando se ejecuta sobre sistema Linux, también se puede ejecutar sobre Solaris y Mac OS. Muchos sistemas que corren sobre Windows utilizan como servidor Web a Apache 2.0 en vez de Microsoft Internet Information Server (IIS).

1.11.4 XSP

XSP es un servidor Web standalone, que al igual que Cassini está escrito en C# y puede ser utilizado para correr aplicaciones ASP.NET con un esfuerzo mínimo (10). El mismo permite hospedar en Linux y otros sistemas operativos UNIX aplicaciones Web implementadas en ASP.NET. Además de ejecutarse sobre la plataforma Mono para Linux, también permite su ejecución sobra la plataforma .NET, posibilitando que sea utilizado como un servidor Web ligero en cualquier plataforma que soporte .NET. XSP fue originalmente el nombre del proyecto de Microsoft que luego se transformó en ASP.NET. El nombre es un homenaje al nombre original de lo que se transformó en ASP.NET. Es completamente código abierto y puede correr en todos los sistemas operativos que soporta Mono (2).

1.12 Compañías que utilizan Mono en la actualidad

Mono actualmente es utilizado por un gran número de vendedores de software comercial y corporaciones con el objetivo de proporcionar un rico ambiente de desarrollo para aplicaciones multiplataforma. A continuación se destacan las compañías más importantes:

Novell

Novell usa Mono tanto para aplicaciones clientes, como aplicaciones servidor. Ejemplo de estas son:

- **iFolder:** es una solución de almacenamiento simple y seguro, que puede aumentar su productividad permitiéndole al usuario realizar copias de respaldo, acceder y administrar sus archivos personales desde cualquier lugar (11).
- **ZenWorks Linux Management:** permite extender fácilmente Linux dentro de un entorno existente (12).
- **Beagle:** producto de búsqueda Desktop.
- **F-Spot:** administrador de fotos.
- **Hula:** servidor Groupware.
- **Banshee:** software reproductor de media.

Metrosharp Corporation

Esta corporación es una especialista en los requisitos de salva de registros complejos y utiliza Mono y Cocoa#, línea de productos *Identity Management System*.

Otee

Su herramienta de modelado para juegos Unity 3D usa Mono para lograr que los usuarios puedan desarrollar juegos multiplataforma.

Versora

Sus especialistas en migración de Windows a Linux usaron Mono y C# para producir una herramienta multiplataforma que le permitiera a la compañía migrar sus sistemas construidos con tecnología .NET de Microsoft, así como, sus configuraciones y datos de usuarios.

Mainsoft

Usa Mono en su producto Grasshopper, cuyo objetivo es permitir desplegar aplicaciones ASP.NET en servidores J2EE.

1.13 Proyectos Web que usan Mono

Existen varios proyectos Web desarrollados con la plataforma Mono, entre los más importantes se encuentran:

Fiducial

Su sitio Web público utiliza Mono y ASP.NET, el sistema corre en un clúster de computadoras con un almacenamiento de datos utilizando PostgreSQL y un sistema administrador de contenido (CMS, por sus siglas en inglés).

EPresence.TV

EPresence Interactive Media es la primera solución para conferencias y webcasting código abierto en el mundo. Este está diseñado para soportar conferencias, reuniones en línea y seminarios difundiéndolos en vivo a través de la Internet, o situándolos disponibles como recurso en demanda.

Wikipedia

Utiliza Mono para mejorar sus facilidades de búsqueda. Su indexado y búsqueda actual está desarrollado en aplicaciones basadas en Mono.

EuroAlert.net

EuroAlert.net (ISSN 1988-3382) es un sitio Web de información gratuita acerca de la Unión Europea que tiene casi 10 años de presencia en la Internet. Gateway S.C.S. ha fusionado todos sus servicios de información acerca de la Unión Europea (Correo de la Unión Europea, Club de Eurogestión y Euroalert) y ha creado un nuevo Euroalert.net con mucha más información y muchos más servicios en los que ha volcado la experiencia acumulada en estos años. Este sitio está construido sobre el marco de trabajo GTW.

Yakugo.com

Es un diccionario Inglés-Japonés Web, basado en AJAX, y que además utiliza Mono en su implementación.

1.14 Conclusiones

En este capítulo se han recogido los principales conceptos necesarios para comprender la tecnología .NET, así como los esfuerzos más importantes realizados en la actualidad para convertirla en una tecnología multiplataforma.

Después de haber analizado los servidores Web más usados, decidimos escoger a Apache como nuestra principal solución para permitir el hospedaje de aplicaciones ASP.NET en Linux; teniendo en cuenta que se necesita un servidor Web que posea características y propiedades que no sólo se asemejen a las de Internet Information Server, sino que de ser posible, tenga un rendimiento mayor, sea escalable y capaz de manejar concurrencia de múltiples usuarios, además de permitir ejecutar aplicaciones ASP.NET. Las características y madurez de Apache 2, asociado a sus prestaciones y escalabilidad exponen por si solas sus potencialidades. De todas las implementaciones multiplataforma de .NET, Mono es la más desarrollada y la que mejores características como entorno de desarrollo y plataforma expone, por tanto decidimos escoger Mono 1.2.6 para trazar nuestra estrategia de portabilidad.

La decisión acerca de que IDE utilizar está determinada por una serie de factores, entre ellos: sistema operativo, costo, facilidad de uso, flexibilidad y funcionalidad, decisión personal, entre otros. Evaluando que los últimos release de Mono permiten cada vez más aprovechar ensamblados desarrollados desde Windows con Visual Studio, sin la necesidad de ser recompilados, además de la alta productividad y el rápido desarrollo de aplicaciones (RAD, por sus siglas en inglés) logrado con este IDE, nuestra estrategia mantiene entre sus principales principios continuar el desarrollo utilizando Visual Studio; en algunos casos, si es necesario realizar algún desarrollo desde Linux, se realizará utilizando MonoDevelop, ya que, es el IDE que presenta mejores características para realizar desarrollo desde la plataforma Linux.

CAPÍTULO 2: IMPLEMENTACIÓN DE .NET DESARROLLADA POR MONO

2.1 Introducción

Con este capítulo, se pretende que el lector conozca las principales características que comprende la implementación de .NET desarrollada por Mono. También se hace referencia a las implementaciones y elementos que incluye Mono que no están comprendidos en la tecnología .NET de Microsoft, y sirven como valor añadido para los desarrolladores de sistemas Unix, Linux y Mac OS X. A través de este capítulo podrá tener una opinión valorativa de lo que es el proyecto Mono, así como, conocer ampliamente los elementos que conforman la plataforma Mono 1.2.6.

2.2 ¿Qué partes implementa Mono?

La implementación de .NET de Mono se divide en dos grupos fundamentales de librerías de clases (conocidos como ensamblados en .NET). Estos dos grupos son: las APIs de compatibilidad con Microsoft y las APIs propias creadas por Mono (13). Los ensamblados de compatibilidad con Microsoft incluyen a ADO.NET para el acceso a datos, ASP.NET para la construcción de aplicaciones Web, y Windows.Forms para el desarrollo de aplicaciones escritorio. Por otro lado, los ensamblados propios de Mono incluyen entre otras cosas el toolkit de interfaces de usuarios GTK#, interfaces para diferentes bases de datos libres, como MySQL y PostgreSQL, y una interfaz para el buscador Web Mozilla. En la siguiente figura puede observar los diferentes componentes que comprenden cada uno de los dos ensamblados.

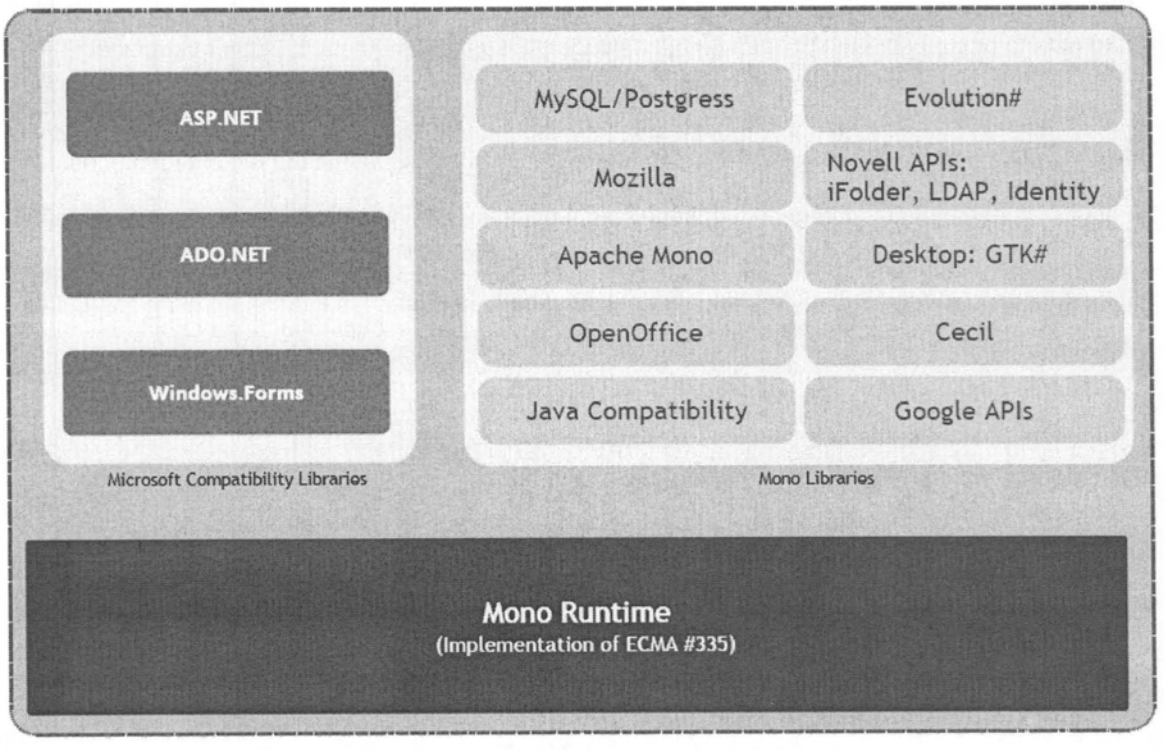

Figura 7: Componentes de los ensamblados implementados por Mono.

No todos los componentes representados en la figura 7 están completamente implementados actualmente. Mono además, desarrolla la implementación basada en C# de varios componentes interesantes. Entre los más importantes se encuentran:

- **GTK#:** Adaptadores (bindings) del popular toolkit GUI GTK+ para sistemas Unix y Windows. También están disponibles otros bindings como son: Diacanvas-Sharp and MrProject.
- **#ZipLib:** Una librería para la manipulación de diferentes ficheros comprimidos y ficheros (zip y tar).
- **Tao Framework:** Bindings para OpenGL.
- **Mono.Directory.LDAP / Novell.Directory.LDAP:** Permiten acceso a través del protocolo LDAP a las aplicaciones .NET.
- **Mono.Data:** Implementación de proveedores para la conexión a distintas fuentes de datos o bases de datos, entre las cuales se encuentran: PostgreSQL, MySQL, Firebird, Sybase ASE, IBM DB2, SQLite, Microsoft SQL Server, Oracle, and ODBC.

- **Mono.Cairo:** Bindings para la máquina de renderizado de gráficos llamada Cairo (el espacio de nombres System.Drawing está implementado basado en esta).
- **Mono.Posix / Mono.UNIX:** Bindings para construir aplicaciones POSIX usando C#.
- **Mono.Remoting.Channels.Unix:** Sócalos (socket) para la conexión remota en sistemas Unix.
- **Mono.Security:** Marco de trabajo para lograr una seguridad y criptografía mejorada.
- **Mono.Math:** Creada para lograr la generación de enteros grandes (BigInteger) y números primos (Prime).
- **Mono.Http:** Soporte para crear servidores HTTP embebidos y personalizados, así como manejadores HTTP par las aplicaciones.
- **Mono.XML:** Soporte extendido para XML.
- **Managed.Windows.Forms (System.Windows.Forms):** Es un System.Drawing completo y multiplataforma basado en Winforms.
- **Remoting.CORBA:** Implementación de CORBA para Mono.
- **Ginzu:** Una implementación sobre Remoting para la pila ICE.

2.3 Principales características de Mono

- **Independencia de lenguaje:** puede utilizar en sus aplicaciones tanto clases implementadas en C#, VB, Java, etc.
- **Independencia de la plataforma:** las aplicaciones construidas con Mono son altamente portables, sus binarios son compatibles con un gran número de plataformas.
- **Amplio soporte para gestores de bases de datos:** MS SQL, MySQL, Postgres, OLE DB, ODBC, etc. En total 27 gestores de bases de datos soportados.
- **Velocidad:** el código IL es compilado por un compilador JIT, esto lo convierte más rápido que lenguajes interpretados como, PHP. El JIT implementado por Mono es un poco más rápido que el implementado por Java, debido a algunas ventajas que posee el código CIL sobre los *bytecodes* de Java. Si es un poco más lento que C (14).
- **Recolección de basura:** con la propiedad de código manejado se puede concentrar mejor en la codificación de su aplicación, sin preocuparse mucho por la liberación de recursos.
- **Seguridad:** las aplicaciones pueden correr dentro de un sandbox que permite la seguridad dentro del contexto de aplicación.
- **Aplicaciones Web:** todos los lenguajes soportados por la plataforma Mono pueden ser utilizados para desarrollar aplicaciones Web.

- **Servicios Web:** Mono brinda soporte para el protocolo SOAP.
- **Soporte para XML:** implementa varias clases que permiten el trabajo con XML.
- **Aplicaciones GUI multiplataforma:** la librería GTK# es bastante poderosa y está implementada para casi todas las plataformas.
- **GNome, KDE y aplicaciones Windows:** Mono tiene bindings completos con las librerías GNome y qt. Además está implementando System.Windows.Forms.

2.3.1 Características del release de Mono 1.2.6

Mono 1.2.6 es una actualización completa del release anterior 1.2.5, perteneciente a la serie Mono 1.2. Este release comprende un arreglo de los principales bugs encontrados en todos los componentes soportados hasta el momento, y también incluye actualizaciones pertenecientes a la pila 2.0. Entre las principales características de este release se encuentran (15):

- Un manejador (driver) de Windows.Forms nativo para Mac OS X, el cual permite correr aplicaciones basadas en Winforms sin la necesidad de X server.
- Soporte para controles y APIs de AJAX para ASP.NET.
- Soporte para despliegue de aplicaciones Web con FastCGI. A partir de ahora ASP.NET puede ser desplegada en una multitud de servidores que soporten el protocolo FastCGI (lighttpd por ejemplo) además del servidor Apache.
- Soporte para Windows.Forms del WebControl en Windows y Linux usando Mozilla.
- El entorno de ejecución consume mucho menos memoria para aplicaciones desarrolladas con el marco de trabajo 2.0, ya que, se realizaron varias optimizaciones para el soporte de la genericidad (generics), además de incluirse varias mejoras de rendimiento, un verificador actualizado y la implementación de seguridad CoreCLR.
- El compilador de C# casi soporta completamente C# 3.0.
- Más de 50 bugs fueron arreglados.
- Mono 1.2.6. ya puede ser usado como SDK para desarrollar aplicaciones Silverlight 1.1 en todas las plataformas que soporta Mono.

2.3.2 Nuevo en Mono 1.2.6

Además de todo lo expuesto anteriormente, el release 1.2.6 presenta nuevas características que mejoran considerablemente la implementación anterior de este marco de trabajo. Entre las más significativas están:

Implementación de AJAX para ASP.NET.

Este release incluye un motor de AJAX para las páginas ASP.NET desarrollado por la compañía Mainsoft que permite mantener la pila de AJAX en el lado del servidor. Esta implementación la puede encontrar en el ensamblado **System.Web.Extensions**. Este motor de AJAX permite correr el toolkit código abierto de ASP.NET (*http://www.asp.net/ajax/ajaxcontroltoolkit/samples/*).

A partir de ahora se encuentra disponible la librería de clases cliente de AJAX para ASP.NET, la cual se encuentra bajo la Licencia Permisiva de Microsoft (MPL: por sus siglas en inglés).

FastCGI

Esta versión de Mono permite realizar el despliegue de aplicaciones ASP.NET en otros servidores que soportan el protocolo FastCGI, además de Apache. Puede visitar el sitio (*http://www.mono-project.com/FastCGI*) si desea obtener más información al respecto.

Windows.Forms

Se ha implementado un nuevo control Web llamado WebBrowser. El mismo brinda soporte para realizar búsquedas básicas, aunque todavía no presenta soporte para realizar manipulación del Modelo de Objetos del Documento (DOM, por sus siglas inglés). Este componente requiere ser utilizado con el navegador Web Mozilla y actualmente sólo funciona en Windows y en Linux.

No se necesita tener un X server instalado en Mac OS X para poder correr algunas aplicaciones escritorio.

Ha sido implementado el contexto de aplicación **WindowsFormsSynchronizationContext**, este permite hacer llamadas a callbacks desde **BackgroundWorker** para correr hilos de interfaces de usuario.

Motor del entorno de ejecución

Esta nueva versión de Mono permite el soporte para el formato de ficheros de ensamblados PE32+. También soporta la localización de memoria manejada: a partir de ahora el entorno de ejecución puede reservar algunas zonas de memoria completamente en código manejado, sin tener la necesidad de hacer una transición a código no manejado. Se redujo significativamente el tiempo de ejecución y la sobrecarga de memoria a la hora de proporcionar interfaces adicionales para los vectores 2.0.

Se implementó la carga perezosa de las interfaces de las tablas de redireccionamiento, mejorando el uso de la memoria. Esta característica está implementada para las plataformas x86, x86-64 y sistemas ARM.

Se implementó el soporte de seguridad CoreCLR para aplicaciones Silverlight. Además de la implementación de un nuevo temporizador (timer), una versión mucho más rápida que usa ThreadPool y proporciona una semántica más adecuada.

Código genérico compartido: los primeros componentes para el código genérico compartido están disponibles a partir de este release. Actualmente puede controlar el nivel de código compartido editando la variable de entorno **MONO_GENERIC_SHARING**.

Soporte para Silverlight

Se han implementado librerías que permiten soportar aplicaciones Silverlight 2.1. Para poder compilar código Silverlight debe utilizar la utilidad `smcs` desde la línea de comandos, esta permite cargar todos los ensamblados pertenecientes a Silverlight y además la versión de lenguaje LINQ (`-langversion:linq`) por defecto.

ASP.NET

Con este release se implementaron nuevas directivas para controlar el tiempo de vida de mod-mono-server. También a partir de ahora el inicio de mod-mono-server esta sincronizado con todos los procesos de Apache.

Puede utilizar una nueva herramienta llamada `mconfig` para extender las configuraciones del fichero **Web.config**, incluyendo nuevas propiedades en el mismo.

El nuevo script `mono-asp-apps` es una herramienta que puede ser usada por distribuciones basadas en RPM, y de esta forma instalar y eliminar aplicaciones ASP.NET de su sistema.

System

En este release se implementó la clase **SslStream**, se mejoró **System.Net.Mail**, se implementó **System.Net.NetworkInformation** (para Windows hasta el momento), se implementó una herramienta basada en Cecil que utiliza reflexión para realizar revisiones de los ensamblados. Por último, se le realizaron mejoras considerables a la sincronización de hilos SSL/TSL. Esto mejora el soporte sobre múltiples hilos de LDAPS.

System.Drawing /Libgdiplus

En esta versión libgdiplus ha sido actualizada para permitir el soporte de las librerias Cairo 1.4.10, las principales actualizaciones son: se añadió la función **GdipCloneFontFamily** y ahora **TextureBrush** soporta mapas de bits transparentes.

Soporte para LDAP

La librería **Novell.Directory.Ldap** ha sido sincronizada con el SDK CSHARP LDAP de Novell (versión 2.1.8). Todos los cambios realizados a partir de ahora se realizarán en conjunto, por lo tanto el nivel de compatibilidad e interoperabilidad será mayor entre las dos.

Compilador de C#

A partir de este release el compilador de C# brinda soporte para métodos parciales y se puede agregar el soporte del lenguaje LINQ utilizando simplemente la opción `-langversion:lin` desde la línea de comandos. Esto se logra a través de la utilidad `smcs` como ya se hizo referencia anteriormente.

2.4 Instalación de Mono

Como se hizo referencia en el capítulo 1, Mono está implementado para un número considerable de plataformas. En este epígrafe conocerá como realizar la instalación de Mono en Linux, específicamente en la distribución Suse Linux Enterprise Server 10 (SLES 10). Ahora, ¿Por qué SLES 10? Primeramente debe conocer las principales características de SLES 10, que lo convierten en una

plataforma con notables propiedades a tener en cuenta a la hora de elegir una plataforma de despliegue para nuestras aplicaciones empresariales.

SLES 10 es un servidor de calidad empresarial diseñado para manejar críticas cargas de trabajo en grandes centros de datos. Desarrollado y mantenido por Novell, esta plataforma servidor ofrece una solución abierta, escalable y de alto rendimiento para centros de datos, exponiendo características para la seguridad de aplicaciones, virtualización, y sistemas de administración integrados presentes en un gran número de arquitecturas hardware. Además, otra de las características que condujo a seleccionar SLES 10 es que al igual que Mono es desarrollado por la compañía Novell, por lo tanto, el nivel de compatibilidad entre ambos es bastante alto.

Una vez escogida la plataforma donde se procederá a instalar Mono, el próximo paso es, seleccionar la versión de la implementación de Mono que instalará, en este caso Mono 1.2.6. Debe conocer que el marco de trabajo de Mono se encuentra disponible desde su sitio oficial en (*http://www.go-mono.com/mono-downloads/download.html*) y aquí puede encontrar RPMs (paquetes ya compilados para plataformas específicas) o puede descargar el código fuente y compilarlo para su distribución.

Si escoge la variante de los paquetes RPM ya compilados, debe tener en cuenta que es muy posible que no todos los paquetes sean compatibles con su distribución, debido a que, pueden necesitar dependencias de otros paquetes que debería tener instalados su sistema previamente, por lo tanto, revise con cuidado la documentación presente en el sitio y solucione tales problemas de dependencias si desea utilizar esta variante, de lo contrario una instalación satisfactoria no podrá ser llevada a cabo.

Para instalar Mono desde RPMs, entonces debe tener en cuenta, realizar las actualizaciones pertinentes para las librerías GLIBC de su sistema. Luego de haber actualizado su sistema con las dependencias necesarias solo tendrá que abrir una consola, situarse en el directorio donde se encuentran los RPMs y como usuario root ejecutar:

```
$ rpm -U *.rpm
```

Una vez ejecutado este comando desde la línea de comandos quedará instalado Mono y todos los demás paquetes de componentes que contiene el marco de trabajo.

La otra variante es realizar la instalación desde los ficheros de código fuente, compilando estos para su distribución específica y evitando tener que actualizar su distribución con paquetes necesitados por los

RPMs. Otra facilidad que brinda realizar la compilación del código fuente es que permite hacer un poco más personalizada la instalación escogiendo prefijos de instalación, así como otras opciones a la hora de compilar el código fuente.

A continuación conocerá los pasos que debe llevar a cabo para compilar el código fuente de Mono. Antes de proceder con la instalación verifique que el sistema tiene instalados: los compiladores de C (gcc, c++ y sus dependencias), también verifique si se encuentra instalado el compilador *bison*, *GNU m4*, el paquete *pkg-config* y el paquete *gettext*, además de las librerías *glib 2.0* o superior.

1. El primer paso que debe realizar es descargar el código fuente de Mono, que se encuentra en (*http://www.go-mono.com/mono-downloads/download.html*).
2. Después de haber realizado la descarga, abra una terminal de su sistema y verifique que se encuentra como usuario con privilegios root.
3. Luego descomprima el fichero del código fuente que descargó con el comando `tar -xvzf mono-1.2.6.tar.gz`, donde, por ejemplo, mono-1.2.6.tar.gz es el nombre del fichero que descargó.
4. Lo siguiente es cambiarse al directorio donde el fichero ha sido descomprimido, para ello utilice el comando `cd`, ejemplo `cd mono-1.2.6` o `cd` seguido por el nombre del directorio donde esta situado el fichero.
5. Ahora, antes de compilar el código debe configurar los ficheros *make*. Para lograr esto, simplemente escriba: `./configure --prefix=/usr` . Le llevará un poco de tiempo realizar toda la configuración.
6. Después ejecute `make` y por último `make install` para realizar la instalación de Mono.

Los pasos descritos anteriormente son bastantes sencillos, a partir de este momento su sistema cuenta con todo un entorno de ejecución que le permitirá desplegar aplicaciones .NET.

2.4.1 Instalar Mono a través de un instalador

Mono Installer 1.2.6 permite una forma fácil de instalar y correr Mono en una variedad de distribuciones Linux. Mono Installer 1.2.6 es un instalador desarrollado por la compañía BitRock, que se dedica a construir software código abierto con mayor facilidad de uso, proporcionando una solución completamente automatizada para el despliegue de aplicaciones código abierto. Los binarios de este instalador funcionan en diferentes distribuciones de Linux.

El primer paso para realizar la instalación utilizando el instalador es darle permisos de ejecución:

```
$ chmod +x mono-1.2.6-linux-installer.bin
```

Después de esto, puede mandarlo a ejecutar dando doble click sobre él o escribiendo el siguiente comando desde una terminal:

```
$ ./mono-1.2.6-linux-installer.bin
```

El instalador también puede ser mandado a ejecutar en modo texto o modo no atendido de la siguiente forma:

```
$ ./mono-1.2.6-linux-installer.bin --mode text

$ ./mono-1.2.6-linux-installer.bin --mode unattended
```

El próximo paso es seguir las pantallas que va presentando el instalador. Por defecto Mono será instalado en el directorio `~/mono-1.2.6`, pero puede cambiar este directorio a preferencia. Después de terminada la instalación un icono será creado en el escritorio de su PC, el cual sirve de acceso directo al directorio donde han sido instalados los diferentes componentes y aplicaciones de Mono, ejemplo, Mono Doc, Monodevelop, etc.

Componentes que instala Mono Installer

- **Mono Runtime:** Este componente está conformado por un compilador del lenguaje C#, una máquina del entorno de ejecución compatible bajo la especificación ECMA-335 y las librerías de clases.
- **ADO.NET:** Incluye soporte para los gestores de bases de datos MySQL, SQLite, IBM DB2, Sybase, PostgreSQL y Oracle.
- **ASP.NET:** Incluye componentes para el desarrollo de aplicaciones Web en ASP.NET e incluye además el servidor Web de pequeña escala XSP.
- **GTK#:** Es una librería para el desarrollo de aplicaciones GUI dirigidas por eventos. Esta librería permite construir aplicaciones completamente nativas en Gnome a través de Mono e incluye soporte para desarrollar interfaces de usuario usando el constructor de interfaces Glade.
- **Mono Develop:** Un IDE para el desarrollo de aplicaciones en Mono.

- **MonoDoc:** Un visor y editor de documentación escalable, desarrollado especialmente para Mono y .NET.
- **IKVM:** Una máquina virtual para el lenguaje Java de .NET.
- **Windows Forms:** Soporte para los release previos de las librerías GUI de Windows System.Windows.Forms.

Una vez instalado Mono debe tener presente que si desea que aplicaciones como MonoDoc y MonoDevelop funcionen correctamente debe revisar la versión de las librerías *gtk-sharp* que tiene su sistema, ya que estas no funcionan de la misma forma para todas las distribuciones. Además para instalar MonoDevelop debe tener instaladas las librerías de Gnome.

Durante la instalación el fichero `bin/.installer_post_libscan` es ejecutado para lograr encontrar las librerías y dependencias necesarias que debe tener su sistema. Esto permite detectar si falta alguna librería y el instalador realizará su notificación una vez terminado el proceso de instalación.

Si selecciona la opción de tener modificado el entorno del sistema automáticamente por el instalador, entonces necesitará reiniciar el shell para que estos cambios puedan tener efecto. Si en cambio decide modificar el entorno usted mismo, entonces debe cargar las configuraciones necesarias ejecutando el fichero `bin/setup.sh`. Esto se logra ejecutando:

```
$ source <install dir>/bin/setup.sh
```

Debe reemplazar `<install dir>` por el directorio que escogió para instalar Mono. Es importante saber que estas configuraciones no son permanentes y solamente afectaran al entorno actual.

2.5 Utilizando la Caché Global de Ensamblados

Como ya conoció en el capítulo 1, los ensamblados son unidades de código que pueden contener un punto de entrada para la ejecución de una aplicación, claro con la excepción de los ensamblados librerías, los cuales son usados simplemente como referencias en otras aplicaciones. Los ensamblados típicamente están ubicados en disco, y el manifiesto indica su número de versión, el cual es resuelto en tiempo de ejecución por el cargador de clases.

La Caché Global de Ensamblados no es más que, un único repositorio en disco de todos los ensamblados que pueden ser referenciados en tiempo de ejecución por las diferentes aplicaciones

construidas por los desarrolladores. El cargador de clases administra toda esta búsqueda automáticamente, buscando en la GAC si es necesario según el orden de búsqueda de ensamblados. Más adelante explicaremos los pasos que son llevados a cabo para localizar un ensamblado.

La GAC es un buen lugar donde se instalan ensamblados comunes para darle la posibilidad de ser usados por una o más aplicaciones. Es importante destacar que el CLR tiene conocimiento de la GAC para buscar en ella ensamblados referenciados, en cambio el compilador no la conoce; por lo tanto el compilador necesita que se le especifique explícitamente la existencia de un determinado ensamblado dentro de la GAC, utilizando para ello el uso de paquetes que son referenciados por el compilador utilizando la opción de la línea de comandos `-pkg`.

2.5.1 Instalar ensamblados dentro de la Caché Global de Ensamblados

Normalmente, un programa ejecutable solo puede localizar ensamblados que se encuentren en el mismo directorio donde está el programa ejecutable. Como pudo ver anteriormente, la GAC permite que los ensamblados estén disponibles para los ejecutables de un sistema determinado, desde un mismo repositorio en disco. Piense en la GAC como si fuera `/usr/lib` en un sistema Linux.

Antes de que un ensamblado pueda ser instalado dentro de la GAC, debe ser firmado para asegurar que el ensamblado hace lo que realmente debe hacer. El primer paso en este proceso es crear un valor hash para el ensamblado utilizando el algoritmo SHA1. Este hash es almacenado en el manifiesto del ensamblado, y utilizado para asegurar que el ensamblado no haya sido dañado o cambiado por alguien sin permiso. En tiempo de corrida, el CLR re-calcula y compara los dos valores, si los dos no coinciden, entonces el CLR no cargará el ensamblado. En función de asegurar que el hash en el manifiesto no pueda ser dañado o modificado, este es encriptado con un nombre seguro (strong name), el cual no es más que un par de clave pública y clave privada.

Puede generar este par de claves utilizando la utilidad `sn`. Las claves con nombre seguro son contenidas dentro de un fichero con la extensión `.snk`. Ejemplo, para generar estas claves debe hacerlo de la siguiente forma:

```
$ sn -k  key.snk
```

Seguidamente, para firmar el ensamblado con esa clave, edite el atributo **AssemblyKeyFile** dentro del fichero **AssemblyInfo.cs** con el siguiente código, por supuesto utilizando el camino relativo al fichero SNK:

```
[ assembly: AssemblyKeyFile("key.snk") ]
```

Ahora, cuando se compile el ensamblado, este va a ser firmado con la clave privada. Una vez firmado el ensamblado, este puede ser instalado en la GAC, utilizando una herramienta construida para este propósito, la **gacutil.exe**. Las opciones soportadas por esta utilidad desde la línea de comandos son:

Bandera	Descripción
`-i`	Instalar un ensamblado dentro de la GAC
`-il`	Instalar una lista de ensamblados dada una lista de nombres de ensamblados
`-l`	Listar el contenido de la GAC
`-u`	Desinstalar un ensamblado de la GAC
`-us`	Desinstalar un ensamblado especificando su nombre
`-ul`	Desinstalar ensamblados dada una lista de nombres de ensamblados

Tabla 1: Opciones soportadas desde la línea de comandos por la utilidad gacutil.

Al instalar el ensamblado, imagine que tenemos un ensamblado llamado *New.dll*, desde la línea de comandos (como root) escriba lo siguiente:

```
$ gacutil -i  New.dll
```

A partir de este momento otros ensamblados pueden hacer referencia al ensamblado *New.dll* sin tener que estar situados en el mismo directorio. Para eliminar ensamblados de la GAC también se utiliza la herramienta gacutil.exe. Ejemplo:

```
$ gacutil -u  New.dll
```

2.5.2 Búsqueda de ensamblados

Para la localización de un ensamblado, el CLR busca en tres lugares diferentes con el objetivo de resolver las referencias necesarias para lograr ejecutar sus ensamblados dependientes. Estos tres lugares son:

- El mismo directorio donde se está ejecutando el ensamblado.
- Los directorios especificados en la variable de entorno `MONO_PATH`, donde cada directorio está separado por dos puntos (:). Usted puede establecer esta variable de entorno con el método apropiado según el sistema operativo que esté utilizando, ejemplo, en Linux puede hacerlo con bash.
- Finalmente la búsqueda se realiza en la GAC.

2.6 Utilidades y herramientas instaladas por Mono

Después de haber instalado Mono, tiene la posibilidad de utilizar una serie de herramientas y utilidades de desarrollo y despliegue que proporciona el entorno de Mono 1.2.6. Entre las más importantes se encuentran:

`disco`

Herramienta implementada por Mono para el Descubrimiento de Servicios Web. Es una herramienta utilizada para realizar el descubrimiento de Servicios Web y devolver los documentos que describen esos Servicios Web.

`gacutil`

Utilidad para la administración de la Caché Global de Ensamblados. Es una herramienta utilizada por los desarrolladores para lograr instalar ensamblados dentro de la GAC del sistema, de esta forma ubicando estos ensamblados disponibles en tiempo de ejecución para todas las aplicaciones que utilicen el entorno de ejecución de Mono.

`ilasm`

El ensamblador de código IL. Este es un ensamblador que prácticamente se pudiera decir que es la copia de la herramienta de Microsoft **ilasm.exe**.

`mcs`

El Mono Compiler Suite. Es el compilador del lenguaje C# que implementa Mono, una implementación de la especificación de lenguaje ECMA-334. Este compilador acepta las mismas opciones desde la línea de comandos que el de C# implementado por Microsoft (**csc.exe**).

`mint`

El intérprete de Mono. Es un intérprete que permite la ejecución de aplicaciones sin la necesidad de haber sido compiladas usando JIT. Las instrucciones de la aplicación son directamente interpretadas a instrucciones x86.

`mono`

El generador de código nativo ECMA-CLI implementado por Mono. Es una implementación del CLI de .NET basada en su especificación ECMA. Puede ser usado tanto para correr aplicaciones .NET, como para correr aplicaciones implementadas bajo la especificación ECMA.

`monodis`

El desensamblador de ensamblados. Es una herramienta que permite desensamblar ensamblados en su respectivo código IL. Permite funcionalidades similares a las implementadas por la herramienta de Microsoft **ildasm.exe**.

`monograph`

Información sobre un ensamblado. Es una herramienta que brinda información acerca de un ensamblado, permitiendo crear un grafo o jerarquía de tipos sobre la información contenida en el ensamblado.

`monop`

Mono Class Outline Viewer, es una herramienta que permite ver toda la interfaz de una clase. A través de ella se puede analizar la firma de todos los miembros de una clase.

`sn`

Utilidad para firmar, verificar y comparar nombres seguros de ensamblados pertenecientes al CLR.

`soapsuds`

Mono's Remoting Proxy Generator. Una herramienta que permite generar documentos WSDL y proxies clientes para servicios remotos.

`wsdl`

Mono's Web Service Proxy Generator. Una herramienta para generar clases proxy que pueden ser usadas para acceder a servicios Web.

`xsd`

Utilidad implementada por Mono que permite generar esquemas o ficheros de clases. Esta utilidad fue construida con el propósito de complementar el soporte para la serialización XML que proporciona Mono.

2.6.1 Opciones configurables del entorno de ejecución de Mono

El entorno de ejecución de Mono es altamente configurable, estas configuraciones pueden ser realizadas utilizando la herramienta `mono`, la cual como vio anteriormente, no es más que la implementación del CLI de .NET basada en las especificaciones ECMA. Este entorno de ejecución contiene un generador de código nativo que se encarga de transformar el código IL a código nativo de la plataforma donde se ejecuta la aplicación.

El generador de código puede operar en dos modos: justo en tiempo de compilación (JIT, por sus siglas en inglés) y antes del tiempo de compilación (AOT, por sus siglas en inglés). Ya que, el código

puede ser cargado automáticamente, es importante destacar que aunque el código sea compilado en modo AOT, el entorno de ejecución y el compilador JIT siempre van a estar presentes.

El entorno de ejecución ofrece un número de opciones de configuración a la hora de correr aplicaciones tanto en modo de desarrollo, como de depuración; y existen además opciones para la prueba y depuración del entorno de ejecución. Entre las más importantes se encuentran:

```
--aot
```

Esta opción es usada para pre-compilar el código IL de un ensamblado determinado a código nativo. El código generado es almacenado dentro de un fichero con extensión **.so**. Este fichero es cargado por el entorno de ejecución una vez que el ensamblado es ejecutado. La configuración del modo AOT es más útil si se realiza combinada con la opción (`-O=all`) y la bandera `-shared`, permitiendo que todas las optimizaciones del generador de código sean activadas y llevadas a cabo durante el proceso de compilación. Sin embargo, algunas de esas optimizaciones no son muy prácticas para el modo de compilación JIT debido a que pueden consumir demasiado tiempo.

A diferencia del marco de trabajo de .NET implementado por Microsoft, AOT no genera código independiente del dominio de aplicación, sino que, genera el mismo código que produciría el modo utilizando el compilador JIT. Normalmente, la mayoría de las aplicaciones utilizan un único dominio, por lo tanto, el modo AOT funciona sin ninguna complicación. En cambio, si desea optimizar el código generado para el uso en múltiples dominios de aplicación, debe usar la opción (`-O=shared`). Tiene que saber que el modo de compilación AOT pre-compila los métodos, pero aun así es requerido el ensamblado original, ya que este contiene los metadatos e información sobre las excepciones, la cual no está presente en el fichero **.so** generado en la pre-compilación.

La pre-compilación no es más que un mecanismo para reducir el tiempo de inicio de una aplicación, incrementar el intercambio de código entre múltiples procesos de Mono y evitar los costos de tiempo de inicio incurridos por una aplicación durante el proceso de compilación JIT.

```
--config nombreFichero
```

Carga el fichero de configuración especificado en *nombreFichero* en vez de los ficheros de configuración por defecto. Los ficheros de configuración por defecto normalmente están ubicados en

los directorios `/etc/mono/config` y `~/.mono/config` o también pueden estar especificados por la variable de entorno **MONO_CONFIG**.

`--desktop`

Permite configurar la máquina virtual para ajustarla mejor cuando van a ser ejecutadas aplicaciones escritorio. Esta opción lo que realiza en sí es configurar el recolector de basura para que evite expandir demasiado el área de memoria del montón, y de esta forma disminuir un poco la actividad del recolector de basura.

`--help, -h`

Despliega instrucciones de ayuda acerca de cómo utilizar la herramienta `mono`.

`--optimize=MODE, -O=MODE`

MODE no es más que una lista de elementos separada por coma, donde se especifica la optimización del entorno de ejecución. Si en cambio desea desactivar alguna opción de optimización, lo debe hacer precediéndola de un signo menos (-). Las optimizaciones implementadas hasta el momento son:

```
all, peephole, branch, inline, cfold, consprop, copyprop, deadce,
linears, cmov, shared, sched, intrins, tailc, loop, fcmov, leaf, aot,
precomp, abcrem, ssapre, sse2
```

Por ejemplo, si deseara activar todas las optimizaciones, menos la que permite eliminar código sin utilizar y la llamada a métodos inline, debería hacerlo de la siguiente forma:

```
-O=all,-deadce,-inline
```

`--runtime=VERSION`

Mono soporta diferentes versiones del entorno de ejecución. La versión depende del programa que está corriendo o del fichero de configuración llamado **program.exe.config**. Esta opción puede ser usada para sobrescribir la auto-detección por defecto de la versión del entorno de ejecución, forzando el uso de una versión del entorno de ejecución diferente. Es importante conocer que esta opción solo puede ser usada para seleccionar una versión del entorno de ejecución compatible, superior a la

versión en que ha sido compilado el programa que se desea correr. Por ejemplo, uno de los usos típicos de esta opción sería, permitir correr una aplicación construida con la versión 1.1 del entorno de ejecución en la versión 2.0. Esto se logra especificando lo siguiente:

```
mono --runtime=v2.0.50727 program.exe
```

`--security, --security=mode`

Permite activar el administrador de seguridad, en la actualidad esta opción es una característica experimental y por defecto está desactivada (`OFF`). Usar *security* sin ningún parámetro, es como si se llamara con el parámetro "cas".

`cas` : Este parámetro le permite a Mono soportar atributos de seguridad declarativos, ejemplo, la ejecución de Code Access Security (CAS) o demandas non-CAS.

`core_clr` : Permite habilitar el sistema de seguridad *core_clr*, típicamente utilizado para aplicaciones Moonlight/Silverlight. Este proporciona un sistema de seguridad más simple que el implementado por `cas`.

`--server`

Configura la máquina virtual para que pueda ejecutar operaciones como servidor.

2.7 Perspectivas futuras de la implementación Mono

Mono es un proyecto que continúa en desarrollo y cada día amplía más sus perspectivas para permitir llevar al ambiente de software libre las potencialidades brindadas por la tecnología .NET de Microsoft. El último release de Mono en la actualidad es el 1.9, el cual es el último release antes de que sea liberado Mono 2.0.

Entre las perspectivas de desarrollo futuro de Mono están, lograr soporte completo para todas las características y librerías de clases presentes en el marco de trabajo 2.0 de Microsoft; porque aunque Mono 1.1.x ya haya realizado la mayoría de estas implementaciones, hasta el momento no existe una garantía plena de que estas se encuentren completamente implementadas y que su funcionamiento sea totalmente correcto (18).

Características planeadas:

1. Soporte completo para las librerías de clases núcleo:
    a. ASP.NET 2.0 (menos soporte para dispositivos móviles).
    b. ADO.NET 2.0.
    c. Windows.Forms 2.0.
2. C# 3.0:
    a. LINQ para objetos.
    b. LINQ para XML.
3. Un compilador para Visual Basic completo.
4. Herramientas para desarrolladores de .NET desde Windows utilizando Mono.
5. Depurador para 1.x y 2.x. (18)

2.8 Conclusiones

A lo largo de este capítulo se expusieron las principales características que presenta la implementación de .NET desarrollada por Mono. A través de ellas se ratifica que esta implementación de .NET es la más completa hasta el momento entre las vistas en el capítulo 1. Ahora, también es necesario saber que aunque Mono ha hecho un gran esfuerzo por seguir los pasos de la implementación de .NET de Microsoft, aún no implementa todas las tecnologías que esta presenta en la actualidad. No obstante, si cuenta con un grupo de características y componentes ya implementados que convierten a la implementación .NET desarrollada por Mono en una alternativa a tener en cuenta para lograr desarrollar aplicaciones con la tecnología .NET de Microsoft multiplataforma, y de esta manera contar con una variante para poder desplegarlas en plataforma Linux.

CAPÍTULO 3: DESARROLLO DE APLICACIONES ASP.NET PORTABLES

3.1 Introducción

Por naturaleza, las aplicaciones Web tienden a ser más portables hacia otras plataformas que las aplicaciones escritorio. La interacción de este tipo de aplicaciones con el usuario es realizada a través del navegador Web, lo que garantiza en la mayoría de los casos una alta portabilidad de la interfaz sin tener que emplear un esfuerzo considerable. En este capítulo podrá conocer los aspectos necesarios a tener en cuenta a la hora de construir aplicaciones ASP.NET portables, es decir, que puedan ser desplegadas con el mínimo esfuerzo en plataforma Linux, siendo compatibles con la implementación de .NET desarrollada por Mono.

3.2 Componentes requeridos para construir una aplicación Web

La tecnología ASP.NET ha sido construida sobre varios conceptos, por lo tanto, esta tecnología depende de varias características que proporciona el marco de trabajo de .NET, así como también, del componente más importante, un servidor Web. Entre los componentes más importantes se encuentran:

- **Servidor Web**: un servidor Web es requerido por dos razones. Primera, el servidor Web es el responsable de aceptar las peticiones de recursos, tales como páginas Web, procesarlas y devolver un resultado al usuario. Segundo, el servidor Web es el encargado de mantener una serie de recursos disponibles para aceptar y procesar cualquier petición de un usuario.
- **CLR**: el CLR es el encargado de brindar un entorno de ejecución para el código .NET y sus librerías asociadas. Toda la pila perteneciente a ASP.NET y todo el código personalizado por los desarrolladores es construido utilizando .NET, estando presente el CLR como actor principal para lograr la compilación y ejecución del código.
- **Marco de trabajo de .NET**: el marco de trabajo de .NET no solamente proporciona la implementación de la tecnología ASP.NET, sino también, las librerías y clases asociadas que son requeridas por el desarrollador para poder construir aplicaciones ASP.NET.
- **Aplicaciones Web o páginas**: finalmente, los componentes mencionados hasta ahora soportan e implementan la tecnología ASP.NET, lo cuales permiten diseñar, desarrollar y correr páginas Web y aplicaciones Web. Por consiguiente, las páginas Web o aplicaciones Web en sí,

son los componentes que se despliegan en el servidor Web, para permitir que el usuario final pueda acceder a ellos y utilizar sus servicios.

3.3 Configuración del servidor Web

Como vio en el epígrafe anterior, uno de los componentes elementales necesarios en la construcción de una aplicación Web es el servidor Web. Para permitir publicar aplicaciones Web ASP.NET con Apache es necesario configurar el módulo específico construido por Mono para este propósito, llamado **Mod_Mono**. El funcionamiento básico de este módulo es pasar todas las peticiones de páginas ASP.NET a un programa externo, llamado mod-mono-server, el cual realmente es el encargado de manejar las peticiones. Esta comunicación entre el módulo de Apache y mod-mono-server se realiza usando un zócalo Unix o un zócalo TCP.

El primer paso a seguir es instalar el servidor Web Apache 2.0 en SLES 10. Una vez instalado Apache debe instalar el módulo mod_mono, para ello abra una terminal con privilegios de usuario root y escriba:

```
$ ./configure --prefix=/usr

$ make

$ make install
```

Después de haber hecho esto, ya tiene instalado mod_mono en su sistema operativo, y estará ubicado en el path `/etc/apache2` un fichero que lleva por nombre *mod_mono.conf*. Su contenido es el siguiente:

```
<IfModule !mod_mono.c>
        LoadModule mono_module /usr/lib/apache2/modules/mod_mono.so
    AddType application/x-asp-net .aspx
        AddType application/x-asp-net .asmx
    AddType application/x-asp-net .ashx
```

```
        AddType application/x-asp-net .asax
        AddType application/x-asp-net .ascx
        AddType application/x-asp-net .soap
        AddType application/x-asp-net .rem
        AddType application/x-asp-net .axd
        AddType application/x-asp-net .cs
        AddType application/x-asp-net .config
        AddType application/x-asp-net .Config
        AddType application/x-asp-net .dll
        DirectoryIndex index.aspx
        DirectoryIndex Default.aspx
        DirectoryIndex default.aspx
</IfModule>
```

Este fichero es generado automáticamente cuando se instala mod_mono.

Lo siguiente es configurar Apache para permitir que redireccione las peticiones ASP.NET y sean atendidas a través del módulo mod_mono. Existen varios mecanismos para realizar esta configuración. A continuación se hace referencia a algunos de ellos.

3.3.1 AutoHosting

En un inicio, mod_mono requería tener que configurar los ficheros de configuración de Apache para cada aplicación que se deseaba desplegar. Entonces el equipo del proyecto Mono comenzó a pensar en un mecanismo para evitar estas configuraciones y convertir más flexible y cómodo el despliegue de

aplicaciones ASP.NET en Apache con mod_mono. De esta forma, surge un mecanismo con la capacidad de auto-configuración, el cual tiene como principal objetivo minimizar la configuración requerida para desplegar aplicaciones ASP.NET en Apache. A este mecanismo lo llamaron **AutoHosting**.

Básicamente consiste en especificarle a Apache que cargue el fichero *mod_mono.conf*. Para ello, simplemente lo que tiene que hacer es incluir en el fichero principal de configuración *httpd.conf* la siguiente línea:

```
Include /etc/apache2/mod_mono.conf
```

El path real donde se encuentra *mod_mono.conf* dependerá del path donde haya sido instalado Apache.

La configuración por defecto de mod_mono está configurada para correr aplicaciones con el entorno de ejecución 1.0 de .NET, para poder correr aplicaciones con el entorno de ejecución 2.0 debe especificar la directriz *MonoServerPath* en el fichero *httpd.conf* de la siguiente forma:

```
MonoServerPath "/usr/bin/mod-mono-server2"
```

Si en cambio, ya tiene una configuración establecida, solamente lo que debe hacer es agregar la directriz:

```
MonoAutoApplication enabled
```

Finalmente, debe copiar su aplicación para el directorio raíz de su servidor Apache y a partir de este momento tendrá disponibles sus aplicaciones Web ASP.NET con Apache. Es recomendable que cree un directorio por cada aplicación que desee publicar.

3.3.2 Configurando Mod_Mono

El mecanismo AutoHosting es una forma fácil de realizar una configuración simple y lograr con el mínimo esfuerzo posible desplegar aplicaciones Web ASP.NET con Apache. Sin embargo, cuando la necesidad es configurar una aplicación empresarial y además tiene experiencia configurando Apache, entonces en este caso el mecanismo AutoHosting no es completamente viable y se requiere de un grupo mayor de configuraciones para lograr que la aplicación sea desplegada.

A continuación conocerá las configuraciones necesarias que debe realizar para tener listo su servidor Web para poder publicar aplicaciones ASP.NET. Inicialmente, debe crear un fichero, ejemplo, *miSitioWebASP.conf*, ubicarlo en el directorio deseado y escribir en él la siguiente configuración:

```
Alias /test "/usr/www/demo"

AddMonoApplications default "/test:/usr/www/demo"

MonoServerPath default /usr/bin/mod-mono-server2

<Location /test>

      SetHandler mono

</Location>
```

Describiendo brevemente la configuración escrita en el fichero *miSitioWebASP.conf*, la primera línea especifica a través de la directriz `Alias` que utilizará el path virtual `/test` para hacer referencia al path físico `/usr/www/demo`. La segunda línea utilizando la directriz `AddMonoApplications` especifica el directorio donde va a estar ubicada la aplicación Web ASP.NET y será en este path físico donde se busquen los recursos de la aplicación cada vez que se realice una petición bajo el path virtual `/test`. La directriz `MonoServerPath` indica la localización (`/usr/bin/mod-mono-server2`) donde se encuentra el servicio de mod-mono-server que atenderá la petición, en este caso para permitir desplegar aplicaciones construidas con el entorno de ejecución 2.0 de .NET es necesario especificar que se atenderán con mod-mono-server2. Por último, dentro de la directriz `<Location ...>` ... `</Location>` se especifica que las peticiones del path virtual `/test` serán manejadas por el entorno de ejecución de Mono.

Después de haber realizado esta configuración debe incluir el fichero *miSitioWebASP.conf,* en su host virtual utilizando la directriz `Include`, ejemplo:

```
Include /etc/apache2/miSitioWebASP.conf
```

Finalmente, deberá incluir el host virtual en el fichero de configuración principal de Apache (*httpd.conf*). Reinicie Apache desde la línea de comandos:

```
/etc/init.d/apache2 restart
```

A partir de este momento tiene configurado todo un servidor de aplicaciones ASP.NET, listo para realizar el despliegue de aplicaciones Web ASP.NET 2.0.

3.4 Configuración de la aplicación Web

EN ASP, toda la información sobre la configuración de aplicaciones Web se almacena en el registro del sistema y la metabase IIS. Esta disposición dificulta la visualización o modificación de la configuración porque, a menudo, las herramientas de administración correctas ni siquiera están instaladas en el servidor. ASP.NET presenta un modelo de configuración totalmente nuevo basado en archivos XML sencillos, escritos en lenguaje natural. Cada aplicación ASP .NET posee su propio archivo **Web.config** que se encuentra en el directorio de su aplicación principal. Desde aquí se controla la configuración personalizada, el comportamiento y la seguridad de las aplicaciones Web.

3.4.1 Configuración de la sección system.web

Dentro del fichero **Web.config**, la sección `<system.web>` es un componente de configuración de la aplicación. Usualmente forma parte del fichero **Web.config** de una aplicación, el cual es usado para configurar los distintos componentes comprendidos dentro del ensamblado **System.Web**.

La sección `<system.web>` puede aparecer dentro de una sección `<configuration>` o dentro de una sección `<location>`. Si en cambio esta sección aparece sola, entonces será aplicada a todas las URLs accedidas por **System.Web**:

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>

    <system.web>

    ..............

    </system.web>

</configuration>
```

Más de un fichero **Web.config** pueden existir en una misma aplicación, se puede tener uno por cada directorio. El orden en que serán cargados estos ficheros **.config** es el siguiente:

- Primero, son cargadas las configuraciones globales especificadas en el fichero **machine.config**.
- Segundo, son cargadas las configuraciones del nivel superior de la raíz de la aplicación. Las configuraciones nuevas sobrescribirán configuraciones viejas.
- Por último son cargadas las configuraciones para las URLs particulares pertenecientes a cada directorio anidado.

Por ejemplo, asumiendo que el directorio virtual de mayor nivel de su aplicación es `/app`, significa que la URL `/app/index.aspx` va cargará las configuraciones especificadas en el fichero **machine.config** mas cualquier configuración existente en el fichero `/app/Web.config`; de esa misma forma `/app/demo/index.aspx` cargará la configuración del fichero **machine.config** mas cualquier configuración existente en `/app/Web.config`, así como la configuración especificada en `/app/demo/Web.config`.

Si lo que desea es aplicar una configuración determinada a un grupo de páginas solamente, entonces lo que debe hacer es anidar la sección `<system.web>` dentro de una sección `<location>`, ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>

    <location path="edit.aspx">

        <system.web>

            <authorization>

                <deny users="?" />

            </authorization>

        </system.web>
```

```
    </location>

</configuration>
```

3.5 Estrategia de portabilidad

La implementación de .NET de Mono actual contiene un núcleo de librerías de desarrollo y una serie de herramientas de desarrollo y despliegue, la mayoría de estas herramientas se ejecutan desde la línea de comandos o son compiladores, por lo tanto no existe un remplazo total de las características que presenta Visual Studio para poder desarrollar aplicaciones ASP.NET nativamente en Linux.

Para plantearse una estrategia de desarrollo que permita obtener aplicaciones con un alto grado de compatibilidad con Linux existen un número de formas que puede adoptar, en dependencia de que tan cómodo se sienta trabajando en Linux, pero es muy importante tener en cuenta la alta productividad lograda desarrollando desde Windows.

3.5.1 Desarrollando en Windows, desplegando en Linux.

Teniendo presente todas las potencialidades expuestas por el IDE Visual Studio y todas las características y propiedades que lo convierten en el IDE de más alta productividad para el desarrollo de aplicaciones Web con la tecnología .NET, se propone continuar desarrollando desde Windows con Visual Studio, ya que, los binarios generados por este IDE tienen un alto grado de compatibilidad con Mono. El grado de compatibilidad en muchos casos es tan alto que solamente sería necesario copiar los binarios y ficheros del proyecto para el servidor Linux donde se va a desplegar la aplicación.

Al iniciar el proceso de construcción de una aplicación ASP.NET con Visual Studio 2005, usted debe tener en consideración un elemento muy importante. Este elemento no es más que el tipo de modelo de proyecto Web que va a utilizar en el desarrollo de su aplicación.

Desde Visual Studio 2005 Microsoft incluyó un nuevo tipo de proyecto Web dentro de las plantillas de proyectos llamado Proyecto Sitio Web (Web Site Project). Este nos es más que una versión radicalmente simplificada de la versión de proyecto más compleja, Proyecto Aplicación Web (Web Application Project). Aunque el modelo de Proyecto Sitio Web es considerablemente simple, presenta algunas limitaciones en el modelo de sistema de ficheros que este posee, por esta razón los

desarrolladores volvieron a demandar el viejo modelo de Proyecto Aplicación Web, y de esta forma Microsoft tuvo que volver a incorporar esta plantilla de proyecto en Visual Studio 2005 Service Pack 1.

Esto significa que a la hora de decidir construir una aplicación Web utilizando Visual Studio 2005 existen dos modelos de proyectos a escoger:

- Proyecto Aplicación Web.
- Proyecto Sitio Web.

Es necesario comprender las principales diferencias que existen entre estos dos tipos de modelos, porque la selección incorrecta de uno de estos modelos puede tener repercusiones serias para el proyecto que desea construir. A continuación se exponen las principales características que comprenden estos dos modelos de proyectos.

Características Proyecto Aplicación Web

- Los proyectos de tipo Proyecto Aplicación Web contienen una estructura de ficheros muy parecida a las demás plantillas de proyectos que permite el IDE.
- Los Proyectos Aplicación Web poseen dentro de su estructura de ficheros un fichero *project*.
- Los Proyectos Aplicación Web son compilados en un ensamblado DLL monolítico; para compilar y depurar las páginas es necesario compilar el proyecto completo cada vez.
- Los Proyectos Aplicación Web son desplegados de una vez en una sola DLL, junto con todos los ficheros de contenido estático necesario para su despliegue.

Características Proyecto Sitio Web

- Los proyectos de tipo Proyecto Sitio Web son un caso especial, ellos no se comportan de la misma forma que cualquier otro tipo de proyectos en Visual Studio.
- No contienen dentro de su estructura de ficheros un fichero *project*.
- Los Proyectos Sitio Web son compilados dinámicamente a nivel de página, esto quiere decir que cada página puede ser compilada y depurada por separado.
- Los Proyectos Sitio Web son desplegados como todo un grupo de ficheros, dando la posibilidad que cualquier fichero pueda ser actualizado independientemente.

Cada uno de estos dos modelos posee su fortaleza y su debilidad. Recomendamos desarrollar aplicaciones Web empresariales muy grandes utilizando el modelo Proyecto Aplicación Web (16), siendo una mejor variante para configurar y distribuir la aplicación.

3.5.2 Principal guía para lograr la portabilidad

Desde la versión 1.1.8 de Mono, surge una nueva funcionalidad, implementada con el sentido de eliminar la sensibilidad a las mayúsculas, presente en los sistemas de archivos de las diferentes plataformas, ejemplo Linux, y los separadores de directorios utilizados para los nombres de archivos.

Esta funcionalidad se llama **IOMap** y es una forma fácil y rápida de lograr la portabilidad de aplicaciones con el menor esfuerzo posible. El principal problema por el que surge esta funcionalidad es debido a que los desarrolladores de Windows generalmente están acostumbrados a trabajar con un sistema de archivos que no es sensible a las mayúsculas, esto significa que les resulta indiferente crear un archivo llamado "mydata" en un lugar y acceder a este a través de "MyData" o "MYDATA", indiferentemente sin ningún tipo de problema.

Otro de los problemas recae en que los desarrolladores de Windows comúnmente introducen el carácter separador de directorios ("\") dentro del código fuente, en vez de usar la clase **Path.Separator** y usar **Path.Combine** para combinar los paths utilizados dentro del código fuente de las aplicaciones. Esto es un problema porque los sistemas Unix utilizan el separador "\" como un componente válido dentro de sus nombres de archivos. Aunque un escaso grupo de aplicaciones utilizan path absolutos utilizando las letras de la unidad de disco, esto es también una problemática para los sistemas Unix, debido a que el carácter (:) representa un nombre de archivo válido en Unix, utilizado para separar varios path, por lo tanto el siguiente path estaría correcto en Unix: "A:\file", indicando que es un nombre de archivo válido dentro del directorio actual.

La solución para las problemáticas planteadas anteriormente a la hora de migrar una aplicación construida en Windows hacia Linux es la siguiente. Normalmente, cuando hacemos este tipo de operación, primeramente necesitamos correr la aplicación en Linux y comprobar que esta funciona como realmente debería funcionar. Pero resolver los problemas de path descritos arriba, incluye un número considerable de iteraciones en aras de realizar todos los cambios necesarios teniendo en cuenta el sistema de archivos de Linux.

Todo este proceso lleva tiempo, porque identificar donde están presentes los errores podría tomar bastante tiempo, así el programa podría fallar varias veces lanzando la excepción **FileNotFound** (generada cuando desde una aplicación se hace referencia a un path que no está presente en el sistema de archivos); al final con un poco de esfuerzo se pueden arreglar todos estos errores y finalmente convertir portable la aplicación, pero consumiría un tiempo considerable detectar y arreglar todos estos errores.

También es preciso conocer que todos estos problemas se podrían solucionar de la forma anterior si tenemos todo el código fuente perteneciente a los componentes que intentamos portar, pero el problema sería mayor si utiliza dentro de su aplicación alguna librería de clases implementada por terceros y no posee su código fuente. En este caso no puede solucionar este tipo de problemas referentes al sistema de archivos por esta vía.

Funcionalidad IOMap

En estos momentos Mono ha implementado una capa de portabilidad que permite eliminar todos esos problemas sin tener que realizar cambios al código fuente. Este nuevo marco de trabajo puede ser habilitado editando la variable de entorno **MONO_IOMAP** con uno de los siguientes valores:

- case: permite que todo el acceso al sistema de ficheros se haga sin tener en cuenta las mayúsculas.
- drive: separa los nombres de las unidades de los nombres de path.
- all: activa tanto a case como a drive.

Además, tiene que saber que si activa cualquiera de las opciones anteriores, también es habilitado el mapeo para los separadores de directorio. Para habilitar IOMap en las aplicaciones ASP.NET publicadas con *mod_mono* debe añadir la siguiente directiva al fichero de configuración principal de Apache:

```
MonoSetEnv  MONO_IOMAP=all
```

Utilizar IOMap puede causar un poco de pérdida de rendimiento en sus aplicaciones, por lo tanto, la mejor estrategia podría ser construir su aplicación portable, teniendo en cuenta las problemáticas del sistema de archivos durante el desarrollo de la aplicación y así no tener la necesidad de arreglar estos problemas utilizando IOMap.

Nombres de path absolutos

No utilice nombres de path absolutos en sus aplicaciones, ya que, estos no funcionan en aplicaciones que se deseen portar hacia Linux. Si dentro de una aplicación necesita localizar algunos recursos, utilice para esto las propias APIs que brinda .NET, por ejemplo, para localizar el fichero de configuración de la aplicación sería de la siguiente forma:

```
AppDomain.CurrentDomain.SetupInformation.ConfigurationFile
```

Platform Invocation (P/Invoque)

El código que contiene P/Invoques, o sea llamadas a librerías nativas de Windows debe ser modificado a la librería nativa de Linux correspondiente, y en algunos casos el código deberá ser refactorizado haciendo llamada a una librería diferente que se encuentre implementada en la plataforma Linux.

Uso del registro

Las clases para trabajo con el registro también son implementadas en Mono para Linux, pero estas sólo son útiles si son utilizadas para referenciar opciones de configuración que use la aplicación. Sin embargo, no brindan ninguna información sobre las configuraciones y propiedades del sistema operativo Linux.

3.6 Análisis de la compatibilidad del código

Aunque desde el inicio de la codificación de la aplicación se haya tenido en cuenta implementar código compatible con la plataforma Mono, que es lo recomendado; siempre es bueno realizar un análisis de la compatibilidad de los ensamblados con Mono. Con este propósito surgen una serie de herramientas que ayudan en la realización de esta tarea. Unas desarrolladas por el proyecto Mono, otras por Microsoft; es importante tenerlas en cuenta, ya que permiten observar cómo están codificados los ensamblados y si es necesario realizarle algún tipo de refactorización para lograr la compatibilidad con la plataforma Mono.

3.6.1 MoMA

El Analizador de Mono para la Migración (MoMA, por sus siglas en inglés), es una herramienta que le ayuda a verificar el nivel de compatibilidad que posee un ensamblado desarrollado en Windows y conocer si puede ser desplegado sin necesidad de ser refactorizado. Esta herramienta le ayuda a comprobar si dentro del código del ensamblado se realizan llamadas a P/Invoque, si existen funcionalidades que aún no han sido desarrolladas por Mono o si desde el código se lanzan excepciones no implementadas por Mono.

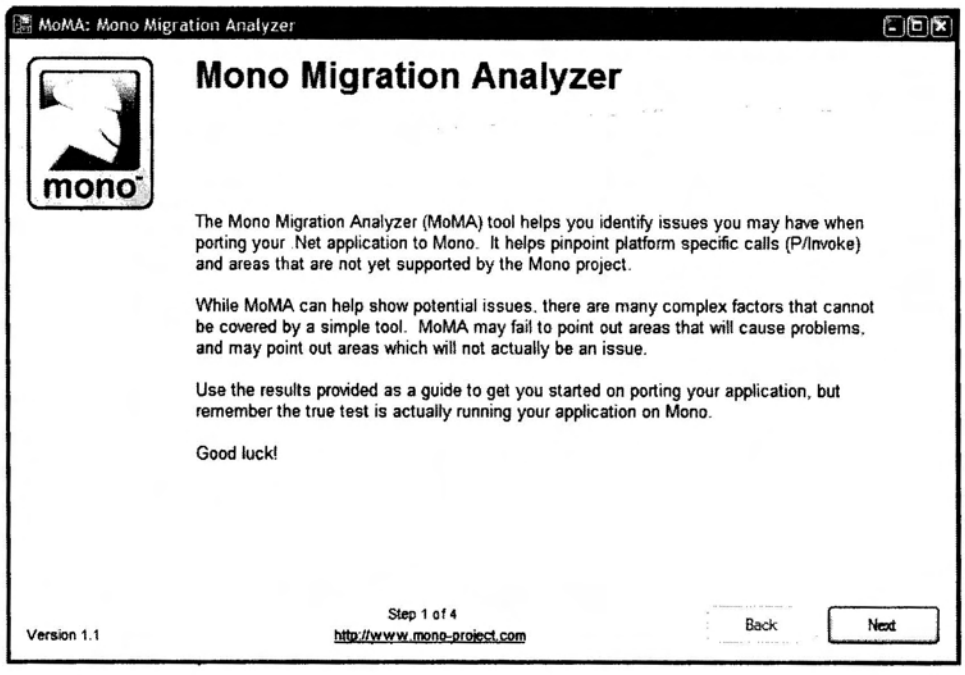

Figura 8: Herramienta MoMA.

Aunque esta herramienta le ayuda a mostrar cuestiones importantes a tener en cuenta a la hora de la migración de una aplicación, tiene que conocer que existen muchos factores más complejos que no pueden ser cubiertos por una simple herramienta. Por lo tanto, MoMA puede fallar algunas veces, no detectando algunas aéreas que pueden causar problemas y en otros casos puede detectar aéreas que realmente no constituyen una amenaza de incompatibilidad. Esto quiere decir que le recomendamos utilizar MoMA como un punto de partida para analizar los ensamblados de la aplicación, ya que la verdadera forma de conocer si su aplicación es 100 % compatible con Mono es corriéndola en Mono.

La versión 1.2.6 de MoMA no es capaz de reportar funcionalidades ausentes en los espacios de nombres **Design**, ya que estas clases solamente son utilizadas cuando se usa un diseñador (designer) como el de Visual Studio y no son utilizadas para correr su aplicación.

3.6.2 Descripción de posibles bugs encontrados por MoMA

Existen 4 tipos de errores e incompatibilidades de ensamblados construidos con Visual Studio desde Windows que pueden ser detectados y reportados por MoMA. A continuación se realiza una descripción detallada de cada uno de ellos, así como, qué hacer para resolverlos.

Métodos Ausentes

Los Métodos Ausentes constituyen el reporte de error más severo. Este tipo de error significa que en el ensamblado existen métodos que hasta la actualidad no han sido implementados por Mono. Si intenta compilar con Mono una aplicación que contiene este tipo de métodos, obtendrá el siguiente error:

```
fichero.cs(22, 16): error CS0117: 'xxxx' does not contain e definition
for 'xxxx'
```

Si en cambio, la compilación es realizada con el compilador de Microsoft, la aplicación correrá en Mono hasta que se haga llamada al Método Ausente. Entonces la aplicación terminará con el siguiente error:

```
System.MissingMethodException: Method not found: 'xxxx'
```

Solución:

El error por llamada a Métodos Ausentes o no implementados puede ser arreglado eliminando del código todos esos métodos antes de que la aplicación sea compilada y corrida en Mono. Alternativamente, la funcionalidad ausente en Mono pudiera ser implementada por el propio desarrollador si realmente es necesaria en para la funcionalidad de la aplicación y enviada a los desarrolladores de Mono para que sea analizada e incluida en versiones posteriores.

MonoTodo

Los métodos marcados como [MonoTodo] pueden causar o no problemas a la hora de la migración de una aplicación. Algunas veces un método puede ser marcado de esta forma para recordarle al

desarrollador que alguna parte pequeña del método no ha sido implementada aún o va a ser mejorada luego. Otras veces, es porque el método no ha sido implementado y simplemente no ejecutará ninguna función al realizar su llamada. Generalmente aunque MoMA detecte métodos marcados como [MonoTodo], la aplicación podrá ser compilada y ejecutada, incluso aunque falte alguna funcionalidad por implementar.

El reporte realizado por MoMA listará una razón específica por la cual el método ha sido marcado como [MonoTodo] (Ver Anexo 1). Además, el grupo de desarrollo de Mono ha pedido a los desarrolladores que si usan [MonoTodo] en algunas de sus implementaciones también deben brindar la razón por la cual están marcados como [MonoTodo] para que sea incluida en el reporte que realiza MoMA. Sin embargo, varias implementaciones en la actualidad no contienen especificada esta razón, es por ello que en algún momento puede encontrar métodos marcados como [MonoTodo] que no digan por qué razón están marcados así.

Solución:

Puede ser que los métodos marcados como [MonoTodo] puedan ser ignorados en un inicio y migrar de todas formas la aplicación, ya que, esta correrá sin ningún problema en Mono. Sin embargo, su aplicación tendrá algunas funcionalidades que no funcionarán. La única solución para los métodos marcados como [MonoTodo] que su funcionalidad no ha sido implementada aún sería trabajar en la implementación de estos métodos, o esperar hasta que el método sea completamente implementado por Mono.

NotImplementedException

Al recibir este tipo de reporte se convierte un poco engorroso saber si existe un error o no, ya que, en muchos casos puede ser que el método realmente no esté implementado y cuando sea llamado ahí mismo lanzará una excepción de tipo NotImplementedException; pero en otros casos puede ocurrir que el método solo lance esta excepción bajo ciertas circunstancias y en la mayoría de las llamadas no lancen la excepción.

Solución:

La solución para este tipo de situación es muy parecida a la de los métodos marcados como [MonoTodo]. Es un poco difícil saber si estos van causar problemas o no. Por lo tanto, podrá compilar

la aplicación y realizarle varias pruebas para saber si realmente necesita cambiar estos métodos o simplemente la aplicación puede funcionar sin la necesidad de arreglarlos.

P/Invoques

P/Invoque es usado para hacer llamadas a funciones que están escritas en leguajes no manejados, algunas veces proporcionadas por la misma plataforma Windows (user32.dll, shel32.dll). Sin embargo, los P/Invoques también pueden ser llamadas a librerías no manejadas construidas por los propios desarrolladores. Mono puede manejar estas llamadas a P/Invoques en los casos en que la librería no manejada se encuentre disponible en la plataforma que usted está usando.

Solución:

Si la aplicación realiza llamadas a librerías que son estrictamente de la plataforma Windows, ejemplo, API win32, entonces debe encontrar una forma de implementar estas funcionalidades en una librería específica para la plataforma de despliegue. Muchas veces esto significa cambiar la llamada a una librería no manejada por una librería manejada equivalente o llamar a la librería de Linux equivalente.

Si por otro lado, la incompatibilidad está presente en una librería nativa implementada por el propio desarrollador, entonces la solución estaría en revisar las capacidades multiplataforma que comprende esta librería y realizar la codificación necesaria para lograr una compatibilidad de la misma con la plataforma de despliegue.

Las llamadas a P/Invoque pueden ser observadas en la última columna que brinda el reporte realizado por la herramienta MoMA, una vez analizado el nivel de compatibilidad de un ensamblado (Ver Anexo 1).

3.6.3 Reflector

La herramienta Reflector es un buscador de clases, explorador, analizador y visor de documentación para .NET. Este permite fácilmente ver, navegar, buscar, descompilar y analizar ensamblados de .NET codificados en C#, IL y Visual Basic. Actualmente se encuentra en la versión 5.1.2.0.

A través de esta herramienta puede desensamblar ensamblados creados con el marco de trabajo de .NET y de esta forma poder revisar si existe alguna implementación en ellos no compatible con la

plataforma Mono, verificando los tipos utilizados en la implementación del ensamblado, las dependencias con otros ensamblados y las llamadas a librerías nativas utilizadas.

A continuación se muestra una figura que muestra la herramienta Reflector.

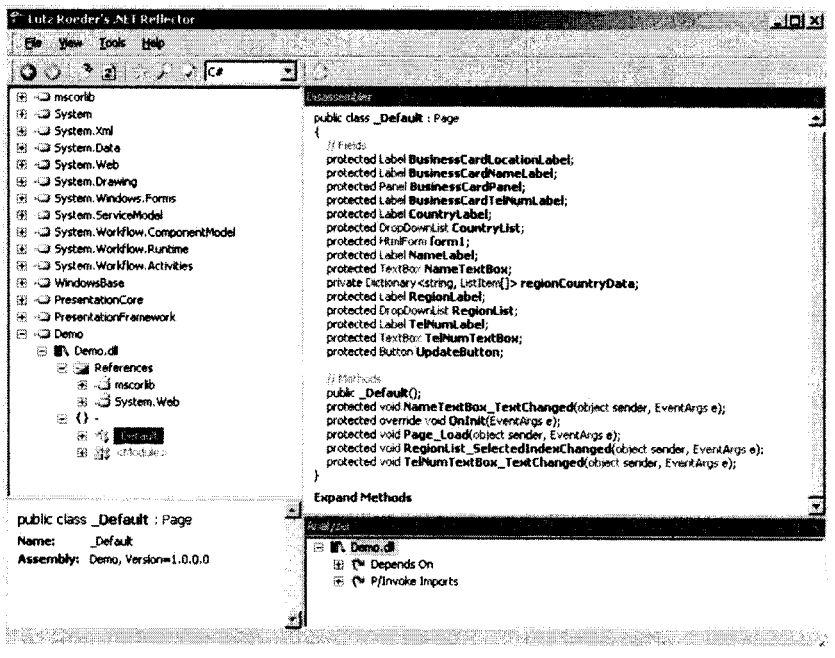

Figura 9: Herramienta Reflector.

3.7 Errores encontrados durante el despliegue

Al realizar el proceso de despliegue de una aplicación Web en Linux es común encontrar ciertas incompatibilidades que necesitan ser resueltas realizando una configuración específica del sistema operativo y de los ficheros de configuración de la aplicación.

3.7.1 DllNotFoundException

Uno de los errores que puede presentar una aplicación a la hora de ser desplegada en Linux es que no encuentre algunas DLLs necesarias para su funcionamiento. A continuación se muestran algunos errores comunes y la forma de darle solución.

```
Server Error in '/val' Application
```

```
gdiplus.dll

Description: Error processing request.

Error Message: HTTP 500. System.DllNotFoundException: gdiplus.dll

Stack Trace:

System.DllNotFoundException: gdiplus.dll

  at (wrapper managed-to-native) System.Drawing.GDIPlus:GdiplusStartup
(ulong&,System.Drawing.GdiplusStartupInput&,System.Drawing.GdiplusStart
upOutput&)    at System.Drawing.GDIPlus..cctor ()
```

Este mensaje de error es causado debido a que Mono no encuentra la DLL (gdiplus.dll).

Microsoft GDI+ (gdiplus.dll) es una librería que contiene funciones básicas relacionadas con el Sistema de Interfaz Gráfico de Windows (GDI, por sus siglas en inglés). Utilizando este sistema Windows es capaz de crear y mostrar objetos en 2D.

Si su aplicación Web está usando librerías de manejo de imágenes, para manipular su tamaño, preservar la calidad al momento de desplegarla utilizando sus dimensiones, etc. Entonces su aplicación realiza llamadas a esta librería.

La solución de Mono para este problema de compatibilidad ha sido la implementación de una librería llamada *libgdiplus*, cuyo objetivo es proporcionar un API GDI+ compatible para otros sistemas operativos diferentes de Windows.

Antes de solucionar el error debe conocer cómo Mono realiza la búsqueda de las librerías necesarias para correr una aplicación. Mono busca las librerías en tres lugares diferentes:

- En el directorio actual.
- En el directorio especificado por la variable de entorno **LD_LIBRARY_PATH**.
- Haciéndole una consulta a la caché del enlazador dinámico del sistema (linker dynamic) utilizando el comando `ldconfig`. Este comando realiza una búsqueda en el fichero `/etc/ld.so.cache`.

Primero, verifique si el fichero correspondiente a la librería *libgdiplus* se encuentra en el sistema, para ello utilice la utilidad `find` de la siguiente forma:

```
$ find /usr -name libgdiplus.so

/usr/local/lib/libgdiplus.so
```

Ahora, ¿Qué es lo que pasa? La librería está ahí, pero Mono no la encuentra. Este problema es porque básicamente por defecto en todas las distribuciones de Linux el enlazador dinámico sólo crea una caché para ficheros ubicados en los directorios `/lib` y `/usr/lib`. Como la librería *libgdiplus* está ubicada en un path diferente el sistema no es capaz de encontrarla. Puede verificar esto de la siguiente forma:

```
$ ldconfig -p |grep libgdiplus
```

El comando no debe producir ninguna salida, porque no se conoce donde está el fichero especificado o puede ser que si muestre una salida, pero indicando que *libgdiplus* se encuentra en un directorio diferente, ejemplo:

```
libgdiplus.so.0 (libc6,x86-64) => /usr/lib64/libgdiplus.so.0

libgdiplus.so (libc6,x86-64) => /usr/lib64/libgdiplus.so
```

Si está referenciado de esta forma es que *libgdiplus* se ha instalado en el sistema por defecto, desinstálela utilizando YAST y haga lo siguiente para solucionar el problema.

La forma correcta para solucionar el error **DllNotFoundException** para lgdiplus.dll es agregando el directorio `/usr/local/lib` como uno de los path que `ldconfig` indexa. Para lograr esto, agregue el path al fichero de configuración `/etc/ld.so.conf` y corra `ldconfig` como root, lo cual forzará la caché para que sea actualizada (rebuilt).

```
$ ldconfig
```

Por último, compruebe la referencia del enlazador dinámico para saber si está apuntando ahora al path correcto.

```
$ ldconfig -p |grep libgdiplus
```

Ahora el resultado debe ser el esperado y el problema con el ensamblado *gdiplus.dll* estará resuelto.

Otra solución podría ser especificar el path donde se encuentra la librería, a través de la variable de entorno **LD_LIBRARY_PATH**, pero no es el procedimiento recomendado, ya que puede causar otros problemas y sería más difícil de mantener.

3.8 Incompatibilidad en los ensamblados de acceso a datos

Cuando se analiza la compatibilidad de una aplicación ASP.NET creada desde Windows es común encontrar cierto nivel de incompatibilidad en los ensamblados que poseen código de acceso a datos, debido a que el acceso a datos desde .NET está implementado a través de la tecnología ADO.NET, la cual no está estandarizada bajo la especificación ECMA. Por lo tanto, es necesario tener en cuenta estas incompatibilidades en aras de refactorizar el código o codificar nuestras aplicaciones compatibles con la plataforma Mono. Entre los errores comunes pueden aparecer:

3.8.1 Llamada al método get_Connections() marcado como [MonoTodo]

La solución para este problema es cargar las configuraciones del fichero **Web.config** a través de *AppSettings*, por lo tanto es necesario cambiar nuestro fichero **Web.config** y poner la cadena de conexión dentro de una sección `<appSettings>`. Después de la modificación el fichero de configuración quedaría de la siguiente forma:

```
...

<configSections>

      <sectionGroup name="mono.data">

            <section                                          name="providers"
type="Mono.Data.ProviderSectionHandler,Mono.Data" />

      </sectionGroup>

</configSections>
```

```
<appSettings>

        <add key="CITeSCS" value="FACTORY=System.Data.OracleClient;DATA
SOURCE=webcitesDB;PERSIST SECURITY INFO=True;USER
ID=WEBCITES;PASSWORD=PASSWORD;UNICODE=True" />

</appSettings>

<!-- Fabrica de providers para la aplicacion. Configuracion del
provider para Oracle-->

<mono.data>

     <providers>

          <provider
name="System.Data.OracleClient"
connection="System.Data.OracleClient.OracleConnection"


adapter="System.Data.OracleClient.OracleDataAdapter"


command="System.Data.OracleClient.OracleCommand"

                              assembly="System.Data.OracleClient,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089"

                              description="Oracle"

               parameterprefix="colon"


commandbuilder="System.Data.OracleClient.OracleCommandBuilder" />

     </providers>
```

```
</mono.data>

...
```

Es muy importante esta configuración porque si las ubicamos dentro de secciones <connectionStrings>, entonces deberían ser cargadas a través del método *get_ConnectionStrings()*, y este es un método marcado como [MonoTodo] para el release de Mono 1.2.6.

3.8.2 Llamada al método ExecuteOracleScalar() marcado como [MonoTodo]

La solución para este problema se realiza cambiando este método por el método *ExecuteScalar()*, el cual realiza la misma operación que *ExecuteOracleScalar()*, devolviendo el mismo resultado y además está completamente implementado en Mono 1.2.6.

```
int value = int.Parse(command.ExecuteScalar().ToString());
```

3.8.3 Llamada al método ausente AddWithValue()

Este error aparece cuando se realiza la asignación a la colección de parámetros de Oracle utilizando el método *AddWithValue()* perteneciente al espacio de nombres **System.Data.OracleClient,** en el ensamblado *System.Data.OracleClient.dll*. Ejemplo:

```
command.Parameters.AddWithValue("PARAM1", username);
```

El problema radica en que *AddWithValue()* es un método que no está implementado aún para la versión 1.2.6 de Mono. Este problema se soluciona realizando la asignación de parámetros a la colección de esta otra forma:

```
command.Parameters.Add(new OracleParameter("PARAM1", username));

command.Parameters.Add(new OracleParameter("PARAM1", password));
```

3.9 Conclusiones

A través del Capítulo se definió toda una estrategia de desarrollo y despliegue, que tiene como principal objetivo contar con una alternativa para lograr que aplicaciones Web construidas en ASP.NET puedan ser compatibles con la implementación de .NET desarrollada por Mono, y de esta forma permitir el despliegue de las mismas en Linux. La estrategia parte desde la implementación y configuración de una plataforma de despliegue para las aplicaciones, hasta el análisis de la compatibilidad que estas poseen con la plataforma Mono.

Aunque Mono es un proyecto bastante avanzado, todavía no existe un entorno de desarrollo que posea una alta productividad para realizar la implementación de aplicaciones Web grandes desde Linux, por lo tanto, la estrategia plantea continuar desarrollando desde Windows con Visual Studio, pero con un alto nivel de compatibilidad con Mono para lograr el despliegue en Linux.

CONCLUSIONES

A partir de la investigación realizada se arriba a las siguientes conclusiones:

- Desplegar aplicaciones ASP.NET en Linux no es una utopía.
- Mono es la implementación portable de .NET que mejores características expone en la actualidad.
- Teniendo bien definida una estrategia de portabilidad se puede lograr desplegar aplicaciones Web en Linux con el menor esfuerzo.
- La estrategia definida constituye una alternativa a tener en cuenta para lograr desplegar aplicaciones ASP.NET en Linux, utilizando Apache 2 y Mono.

RECOMENDACIONES

- Tener en cuenta los aspectos descritos en la investigación a la hora de construir aplicaciones ASP.NET compatibles con Mono.
- Continuar el estudio de la plataforma Mono.
- Realizar la implementación de herramientas personalizadas que ayuden en el proceso de análisis de la compatibilidad con Mono del código de aplicaciones Web implementadas con la tecnología ASP.NET.

REFERENCIAS BIBLIOGRÁFICAS

1. **Nantz, Brian.** *Open Source .NET Development.* New York : Prentice Hall PTR, 2004. 0-321-22810-3.

2. **Thai, Thuan and Lam, Hoang Q.** *.NET Framework Essential, First Edition.* New York : O'Reilly, 2001. 0-596-00505-9.

3. **Stutz, David, Neward, Ted and Schilling, Geoff.** *Shared Source CLI Essentials.* New York : O'Reilly, 2002.

4. **Omnicore.** X-Develop. [Online] Omnicore, 2007. [Cited: 1 20, 2008.] http://www.omnicore.com/en/xdevelop.htm.

5. **Microsoft.** Internet Information Server. [Online] Microsoft, 2007. [Cited: 02 15, 2008.] http://www.microsoft.com/spain/windowsserver2003/technologies/webapp/iis.mspx.

6. **Kementeus.** Cassini Personal Web Server. [Online] My kemen world, 2007. [Cited: 02 15, 2008.] http://kementeus.wordpress.com/2007/02/16/cassini-personal-web-server/.

7. **Microsoft Corporation.** Cassini Sample Web Server. [Online] Microsoft, 2008. [Cited: 02 10, 2008.] http://www.asp.net/downloads/archived/cassini/.

8. **Netcraft.** March 2008 Web Server Survey. [Online] Netcraft, 3 26, 2008. [Cited: 3 28, 2008.] http://news.netcraft.com/archives/web_server_survey.html.

9. **Kabir, Mohammed J.** *La Biblia del Servidor Apache.* Madrid : Anaya Multimedia, 2003.

10. **Dumbill, Edd and Bornstein, Niel M.** *Mono: A Developer's Notebook.* New York : O'Reilly, 2004. 0-596-00792-2.

11. **Novell.** Mono-project. [Online] Mono, 2008. [Cited: 04 05, 2008.] http://www.ifolder.com/index.php/Main_Page.

12. —. Novell ZENworks Linux Management. [Online] Novell, 2008. [Cited: 4 5, 2008.] http://www.novell.com/products/zenworks/linuxmanagement/.

13. **Mamone, Mark.** *Practical Mono.* New York : Apress, 2006. 1-59059-548-3.

14. **Seoane, Fabian.** LUGCIX (Linux User Group Chiclayo). [Online] Linux User Group - Perú, 2004. [Cited: 06 28, 2008.] http://www.lugcix.org/tutoriales/mono/queesmono.php.

15. **Novell.** Mono-progect. [Online] Mono, 2007. [Cited: 05 10, 2008.] http://www.go-mono.com/archive/1.2.6/.

16. **Allen, Scott, et al.** *The ASP.NET 2.0 Anthology: 101 Essential Tips, Tricks & Hacks.* United States of America : SitePoint Pty. Ltd., 2007. 978-0-9802858-1-9.

17. **Novell.** Mono-project. [Online] Mono. [Cited: 03 25, 2008.] http://www.mono-project.com/ASP.NET.

18. —. Mono-project. [Online] Mono. [Cited: 4 3, 2008.] http://www.mono-project.com/FAQ:_General.

19. —. Mono-project. [Online] 2008. [Cited: 5 25, 2008.] http://mono-project.com/Roadmap.

BIBLIOGRAFÍA

1. **Hamilton, Bill.** *ADO.NET Cookbook.* New York : O'Reilley, 2003. 0-596-00439-7.

2. **Bowen, Rich y Coar, Ken.** *Apache Cookbook.* New York : O'Reilly, 2003. 0-596-00191-6.

3. **Ahmed, Mesbah, y otros.** *ASP.NET Web Developer's Guide.* United States of America : Svngress Publishing, Inc., 2002. 1-928994-51-2.

4. **Fedorov, Alexei.** *A Programmer's Guide to .NET.* New York : Addison Wesley, 2002. 0-321-11232-6.

5. **Kabir, Mohammed J.** *Apache Server 2 Bible.* New York : Hungry Minds, Inc., 2002. 0-7645-4821-2.

6. **Riordan, Rebecca M.** *Microsoft ADO .NET Step by Step.* New York : Microsoft Press, 2002. 0735612366.

1. **Hamilton, Bill.** *ADO.NET Cookbook.* New York : O'Reilley, 2003. 0-596-00439-7.

2. **Bowen, Rich y Coar, Ken.** *Apache Cookbook.* New York : O'Reilly, 2003. 0-596-00191-6.

3. **Ahmed, Mesbah, y otros.** *ASP.NET Web Developer's Guide.* United States of America : Syngress Publishing, Inc., 2002. 1-928994-51-2.

4. **Fedorov, Alexei.** *A Programmer's Guide to .NET.* New York : Addison Wesley, 2002. 0-321-11232-6.

5. **Kabir, Mohammed J.** *Apache Server 2 Bible.* New York : Hungry Minds, Inc., 2002. 0-7645-4821-2.

6. **Riordan, Rebecca M.** *Microsoft ADO .NET Step by Step.* New York : Microsoft Press, 2002. 0735612366.

7. **Esposito, Dino.** *Building Web Solutions with ASP.NET and ADO.NET.* New York : Microsoft Press, 2002. 0-7356-1578-0.

8. **Reilly, Douglas J.** *Designing Microsoft ASP.NET Applications.* New York : Microsoft Press, 2002. 0735613486.

9. **Box, Don y Sells, Chris.** *Essential .NET, Volume 1: The Common Language Runtime.* New York : Addison Wesley, 2002. 0-201-73411-7.

10. **Parihar, Mridula y al, et.** *ASP.NET Bible.* New York : Hungry Minds, 2002. 0764548166.

11. **Turtschi, Adrian, y otros.** *C# .NET Web Developer's Guide.* New York : Syngress Publishing, Inc., 2002. 1-928994-50-4.

12. **Walther, Stephen.** *ASP.NET Unleashed, Second Edition.* New York : Sams Publishing, 2003. 0-672-32542-X.

13. **Mitchell, Scott, y otros.** *ASP.NET: Tips, Tutorials, and Code.* Indianapolis, Indiana : SAMS, 2001. 0-672-32143-2.

14. **Microsoft ACE Team.** *Performance Testing Microsoft .NET Web applications.* Redmond, Washingto : Microsoft Press, 2003. 0-7356-1538-.

15. **GRIMES, FERGAL.** *Microsoft .NET for Programmers.* United States of America : Manning Publications Co., 2003. 1-930110-19-7.

16. **Butler, Jason y Caudill, Tony.** *ASP.NET Database Programming Weekend Crash Course.* New York : Hungry Minds, Inc., 2002. 0-7645-4830-1.

ANEXOS

Anexo 1

Reporte realizado por la herramienta MoMA.

DemoApp.exe

Methods missing from Mono

Calling Method	Method not yet in Mono
Class DemoApp.Form1:	
void button1_Click(Object, EventArgs)	void ButtonBase.set_TextImageRelation(TextImageRelation)

P/Invokes into native code

Calling Method	P/Invoke Method	External DLL
Class DemoApp.Form1:		
void button1_Click(Object, EventArgs)	bool SetProcessDPIAware()	user32.dll

Methods called that throw NotImplementedException

Calling Method	Mono method that throws NotImplementedException
Class DemoApp.Form1:	
void button1_Click(Object, EventArgs)	Object Image.Clone()

Methods called marked with [MonoTodo]

Calling Method	Method with [MonoTodo]	Reason
Class DemoApp.Form1:		
void button1_Click(Object, EventArgs)	void RichTextBox.Redo()	Not Specified

GLOSARIO DE TÉRMINOS Y SIGLAS

A

AOT: Ahead-Of-Time

ARM: Familia de microprocesadores RISC diseñados por la empresa Acorn Computers y desarrollados por Advanced RISC Machines Ltd.

C

CAS: Code Access Security

CIL: Common Intermediate Language

CLR: Common Language Runtime

CLS: Common Language Specification

CMS: Content Management System

COFF: Common Object File Format

COM: Component Object Model

CTS: Common Type System

D

DOM: Document Object Model

E

ECMA: European Computer Manufacturers Association

F

FCL: Framework Class Library

G

GAC: Global Assembly Cache

GPL: General Public License

GUI: Graphic User Interface

H

HTML: Hypertext Markup Language

HTTP: Hypertext Transfer Protocol

I

IDE: Integrated Development Environment

IDL: Interface Definition Language o Interface Description Language

ISO: International Standard Organization

J

JIT: Just-In-Time Compilation

L

LGPL: Less General Public License

M

MDE: Model Driven Engineering

MPL: Microsoft Permissive License

MSIL: Microsoft Intermediate Language

P

PAL: Platform Adaptation Layer

PDA: Personal Digital Assistant

PE: Portable Executable

R

RAD: Rapid Application Development

S

SDK: Software Development Kit

SEH: Structured Exception Handling

V

VES: Virtual Execution System

W

WSDL: Web Services Description Language

X

XML: Extended Markup Language