Análisis Comparativo entre .Net y Mono

Mares Valenzuela Obed, Villavicencio Espinoza Ari, Delgadillo Quezada Omar, Ibáñez Salas Juan Francisco y Díaz-Ramírez Arnoldo

***Resumen*-- El desarrollo de aplicaciones corporativas ha evolucionado significativamente en fechas recientes. La gran diversidad de plataformas de hardware, así como de sistemas operativos y herramientas de desarrollo, dificultan la implementación de sistemas modernos. Adicionalmente, sistemas heterogéneos deben interconectarse y trabajar de manera orquestada. En este contexto, el marco de referencia .Net proporciona una alternativa interesante. Sin embargo, tiene como desventaja estar vinculada al uso de herramientas propietarias. Una opción atractiva es el uso de Mono, una plataforma para el desarrollo de aplicaciones con el marco de referencia de .Net, pero con la ventaja de que permite la utilización de herramientas de software libre. En este artículo se presentan los resultados de la comparación entre estas dos plataformas de desarrollo de aplicaciones distribuidas. Se presentan las principales características de .Net y Mono, así como los resultados de la evaluación del desempeño que se hizo entre ellas.**

***Palabras clave*-- Análisis de Desempeño, Desarrollo de Aplicaciones, Microsoft .Net, Mono, Sistemas Distribuidos**

I. INTRODUCCIÓN

EL área de desarrollo de aplicaciones está en un constante cambio y se encuentra encaminada hacia un nuevo esquema de trabajo, donde existen una gran cantidad de usuarios con necesidad de acceso a las aplicaciones distribuidas, con dispositivos más sofisticados. Ya no se limita al uso de la red y la computadora personal, sino que también los

usuarios cuentan con dispositivos móviles tales como teléfonos inteligentes, tabletas, entre otros. Además, las aplicaciones se ejecutan en distintas plataformas (Internet,

nube, etc.), donde se exigen nuevas herramientas para facilitar el diseño y el acceso a bases de datos. En este panorama, el uso de un marco de referencia donde el desarrollador solo se centre en la lógica del programa y que incluya acceso multiplataforma, representa una gran ventaja. La evolución en el desarrollo de aplicaciones y la necesidad de interconectarlas provocó que las herramientas ya existentes se hicieran inadecuadas para la industria del desarrollo de software. En 2005 surge un marco de referencia que cumple con las necesidades de las aplicaciones empresariales, y otorga a los desarrolladores facilidad corrigiendo deficiencias como son los continuos

parches. Este marco de referencia, desarrollado por Microsoft y denominado .Net, crea una amalgama de tecnologías relacionadas, permitiendo que los programadores cuenten con recursos estandarizados y evite la necesidad de mantenimiento a la compatibilidad a versiones previas, debido en parte a un continuo cambio de las interfaces de programación [1].
Un inconveniente es que .Net utiliza software propietario, por lo que una alternativa es la plataforma Mono, que sigue las reglas de .Net, sin dejar de ser software libre. En este artículo se presenta un análisis comparativo entre estas dos plataformas de desarrollo. El objetivo es conocer ventajas y desventajas de cada una de ellas, y de analizar los resultados de un comparativo del desempeño de estas plataformas en un caso experimental. El resto del documento está organizado de la siguiente manera. En la Sección II se presenta el marco de referencia .Net. En la Sección III se explica la plataforma Mono. El análisis comparativo de los experimentos para evaluar estas plataformas se discute en la Sección IV. Finalmente, las conclusiones y el trabajo futuro se explican en la Sección V.

II. MICROSOFT .NET

La plataforma de *Microsoft* para el desarrollo, despliegue y ejecución de aplicaciones, está orientada a servicios sobre entornos altamente distribuidos. Introduce el concepto de los servicios Web, que permite el desarrollo de aplicaciones acopladas basadas en componentes y que utilizan protocolos de comunicación estándar de Internet, como *SOAP* (*Simple Object Access Protoco*l) y *XML* (*Extensible Markup Language*) [2].
Esta plataforma incluye dos partes principales. Una es el *Common Language Runtime* (*CLR*) y la otra es la Biblioteca de Clases del Marco de Referencia. El *Common Language Runtime* (*CLR*) es el responsable de asegurar que el código es ejecutado como se requiere, proporcionando una serie de facilidades para el código *CIL* (*Common Intermediate Lenguage*) [3], tales como:

- Cargar el código y verificarlo
- Gestión de excepciones

Gabriel Obed Mares Valenzuela es estudiante de Ingeniería en Sistemas Computacionales del Instituto Tecnológico de Mexicali, Baja California, e-mail: obed.mares@gmail.com.

- Compilación *Just In Time* (*JIT*)
- Gestión de la memoria
- Seguridad

La biblioteca de clases del marco de referencia incluye un conjunto de paquetes de gran funcionalidad, que los desarrolladores pueden usar para extender más rápidamente las capacidades de su propio software. La biblioteca incluye tres componentes claves:

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

Figura 1. Marco de referencia .Net

Las especificaciones comunes de lenguaje (*CLS*, *Common Lenguage Specification*), son una serie de reglas básicas requeridas para la integración del lenguaje, que garantizan que el código intermedio generado por cada uno de ellos sea interoperable con los otros. Hasta ahora hay una serie Las especificaciones comunes de lenguaje (*CLS*, *Common Lenguage Specification*), son una serie de reglas básicas requeridas para la integración del lenguaje, que garantizan que el código intermedio generado por cada uno de ellos sea interoperable con los otros. Hasta ahora hay una serie de lenguajes propios de *Microsoft*:

- J#
- VB.Net
- Perl.Net
- Python.Net
- C#

El lenguaje C# es probablemente el más popular, con una gran similitud al lenguaje Java [4]. Las principales diferencias de C# con respecto a los otros lenguajes son:

- La gestión de memoria automática
- No es necesario la utilización de punteros; sin embargo es posible utilizarlos
- Se cambia el uso de algunos operadores
- El método *Main* se declara de forma distinta
- No se utilizan archivos de cabeceras *.h* ni similares
- No existe biblioteca de tiempo de ejecución [5]

Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones *Web* ASP.NET, servicios *Web XML*, aplicaciones de escritorio y aplicaciones móviles. *Visual Basic*, *Visual C++*, *Visual C#* y *Visual J#*. Utilizan el mismo entorno de desarrollo integrado (*IDE*), que les permite compartir herramientas y facilita la creación de soluciones en varios lenguajes. Así mismo, dichos lenguajes aprovechan las funciones de .NET *Framework*, que ofrece acceso a tecnologías clave para simplificar el desarrollo de aplicaciones *Web* ASP y Servicios *Web XML*. Brindando las herramientas necesarias para crear, distribuir, administrar y dar mantenimiento a aplicaciones *Web* distribuidas como de escritorio, todo esto con una gran facilidad y rapidez. En la Fig. 2 se muestra un ejemplo de la interfaz de Visual Studio.

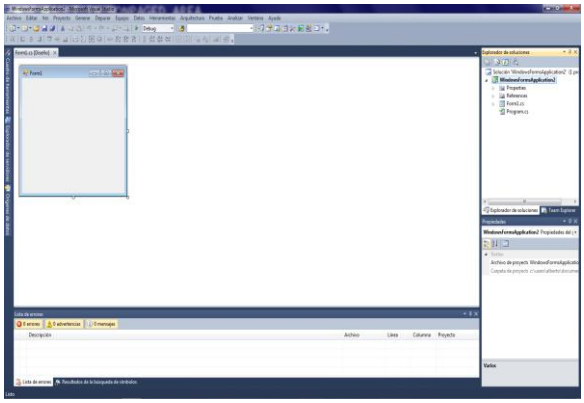

Figura 2. Ejemplo de interfaz de Visual Studio

III. MONO

Es un proyecto *Open Source* iniciado por Miguel de Icaza en 2001, en cual se pretenden crear aplicaciones de manera más rápida en el entorno Linux. Incluye las bibliotecas del .Net *Framework*, pero además cuenta con una serie de bibliotecas no existentes en él, como el *GTK#*, que permite crear interfaces gráficas nativas del *toolkit GTK+*, *Mono.LDAP*, *Mono.Posix*, etc.

Mono es la implementación libre del *CLI* (*Common Lenguage Infraestructure*) la cual contiene la máquina virtual, cuya función consiste en cargar las clases, el compilador *JIT* (*Just-in-time*) y el *garbage collector*. Todo esto escrito desde cero de acuerdo a las especificaciones ECMA-334 [6], incluyendo su compilador, el cual paradójicamente está escrito en C# y al igual que el *CLI*, este compilador sigue las especificaciones ECMA-334.

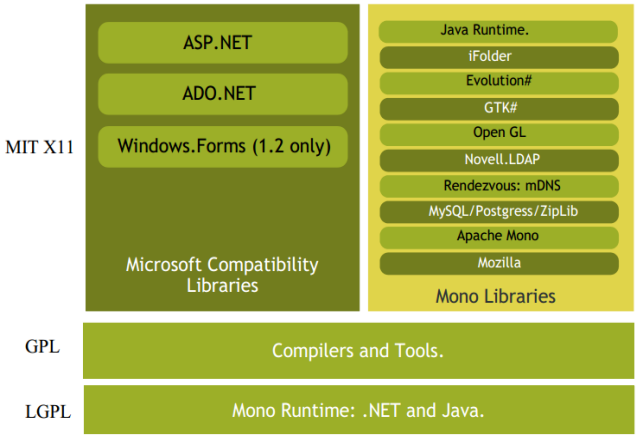

Figura 3. Marco de Referencia Mono

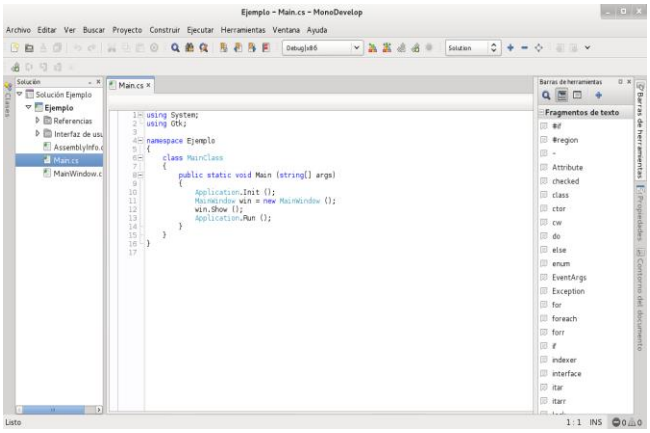

Figura 4. Ejemplo de interfaz de *MonoDevelop*

Mono ofrece todo su código, compiladores, *runtime*, biblioteca de clases, etc., con licencias libres, perfectas para poder aprender sin preocuparse de problemas legales y con la posibilidad de realizar modificaciones en el código para colaborar o distribuirlo ya sea comercialmente o de forma gratuita.

MonoDevelop es un entorno de desarrollo diseñado principalmente para C# y otros lenguajes .NET. Permite a los desarrolladores escribir rápidamente aplicaciones de escritorio y *Web* ASP.NET en Linux, Windows y Mac OSX. *MonoDevelop* hace que sea fácil para los desarrolladores de .NET crear aplicaciones con un entorno de desarrollo similar al *Visual Studio* en Linux, y mantener una base de código único para todas las plataformas. En la Fig. 4 puede apreciarse un ejemplo de la interfaz de *MonoDevelop*. Sus principales características son las siguientes:

- Multi-plataforma compatible con Linux, Windows y Mac OSX
- Edición avanzada de texto: Código de finalización de soporte para *C# 3*, plantillas de código, plegado de código
- Banco de trabajo configurable: Son totalmente configurables, diseños de ventana, atajos de teclado definidos por el usuario y herramientas externas
- Soporte de múltiples lenguajes: *C#*, *Visual Basic.Net*, *C / C + +*, etc.
- Depurador integrado para Mono y nativas
- *GTK # Visual Designer*: Permite crear fácilmente aplicaciones GTK#
- ASP.NET Crea proyectos *web* con soporte completo, completado de código y prueba en XSP, el servidor *web* Mono
- Otras herramientas: Control de código fuente, integración *de makefile*, pruebas unitarias, *packaging and deployment*, localización [7].

IV. ANÁLISIS COMPARATIVO

Para evaluar las características, ventajas y desventajas de ambas plataformas, en primer lugar se evaluaron aspectos cualitativos. Para este fin, se desarrollaron aplicaciones utilizando .Net y Mono y se analizaron los resultados. En la Tabla I se muestra el resultado de este análisis comparativo.

Además de evaluar y comparar los entornos de desarrollo de .Net y Mono, se definieron un conjunto de experimentos para llevar a cabo un análisis comparativo de rendimiento. Con este fin, se desarrolló una aplicación que se conecta a una base de datos, y que hace operaciones de inserción, eliminación y consulta. El objetivo consistió en comparar los tiempos de ejecución de ambas plataformas, para llevar a cabo estas operaciones.

Criterio de evaluación	.Net	Mono
Dificultad de instalación	media	baja
Dificultad en el uso de interfaz de usuario	baja	baja
Dificultad para la creación de una nueva solución	baja	baja
Dificultad en acceso a bases de datos	baja	media
Dificultad en diseño de interfaz	baja	baja
Costo	alto	bajo

Tabla I
COMPARATIVO ENTRE LAS TECNOLOGÍAS .NET Y MONO

Para el análisis comparativo de las aplicaciones, se midieron los tiempos de ejecución para conectarse a la base de datos, insertar, eliminar y mostrar datos. Se utilizó un método de granularidad, que es la parte del código que puede ser

medido, y por lo general se especifica de manera subjetiva. Generalmente se mide el tiempo de ejecución en un proceso por procedimiento, o por función de base [8]. Para ello, se realizaron unos los siguientes cálculos:

$$Tt=(Tf - Ti) \quad (1)$$

en donde:
Tt = tiempo total, *Tf* =tiempo final y *Ti*=tiempo inicial.

Se crearon dos aplicaciones, una en Windows y otra en Linux, utilizando el lenguaje C#. Estas aplicaciones se construyeron de la manera más parecida posible, dado que hay algunas diferencias tanto en los entornos de desarrollo, en la sintaxis y en el administrador de base de datos. Se llevaron a cabo cien experimentos de cada operación y se calculó el tiempo promedio de ejecución en cada caso. Las aplicaciones se invocaron desde consola y se ejecutaron incluyendo una interfaz gráfica, ya que es la manera en que habitualmente se utilizan estas aplicaciones y era importante evaluar su desempeño en un escenario real. A continuación se discuten algunos de los resultados importantes de esta serie de experimentos.

IV-A. ¿Qué diferencias hay de una aplicación a otra?

Las diferencias más relevantes en las aplicaciones son, que la que está desarrollada en *VisualStudio* utiliza *forms*, lo que le da la característica apariencia de Windows. Por otra parte, la que se desarrolló en *Mono Develop* se desarrolló utilizando *GTK#*, que es el API nativo del mismo. Se decidió que fuera de esa forma dado que se obtendría una comparativa más exacta, ya que si se puede usar *forms* en *MonoDevelop*. El administrador de base de datos utilizado en cada plataforma fue distinto. En Windows se usó *SQLserver*, el cual está incluido en el entorno de desarrollo *Visual Studio*, mientras que en *Mono Develop* se usó *MySql*.

IV-B. ¿Cuáles fueron los problemas y como se resolvieron?

El principal problema que se encontró fue que no se podían obtener tiempos de ejecución exactos, ya que la ejecución de la aplicación .Net se hace utilizando una máquina virtual (*Common Lenguage Runtime*). Este dificulta el cálculo de los tiempos de ejecución dado que las instrucciones del programa NO se ejecutan en secuencia, sino desordenadamente. Esto es así porque los modernos procesadores siguen el modelo fuera de orden (OOO:*Out Of Order*). El tiempo de ejecución, para cualquier programa, NO es constante (esta es una característica indeseable para algunos sistemas de tiempo-real, donde es conveniente que el tiempo de respuesta esté acotado). Los datos y las instrucciones se encuentran en memoria principal, pero cuando se ejecutan por primera vez son traídas cerca del procesador, a la memoria caché.

Tiempo(ms)	Conectar	Insertar	Consultar	Eliminar
Microsoft .Net	0.0293	0.0983	0.1675	0.0718
Mono	0.0281	0.0948	0.1599	0.0625
Diferencia	3.84 %	3.52 %	4.49 %	12.94 %

Tabla II
RESULTADOS DE LOS EXPERIMENTOS

En un procesador moderno hay varias jerarquías de memoria caché, y el llenado y vaciado de estas estructuras hacen variable el tiempo de ejecución. Cuando se ejecutan los programas NO se encuentran completamente en memoria principal. Cuando hay que acceder al disco a buscar el resto del programa, el sistema operativo detiene la ejecución del programa. Con base en esto se evaluaron tres posibles soluciones, las cuales fueron:

- Utilizar un sistema operativo de tiempo-real para hacer dichos experimentos. Sin duda alguna era la forma más precisa, pero fue descartada debido al costo de estos sistemas o su poca disponibilidad, específicamente en el caso de Windows de tiempo-real. Además de que .NET no está diseñado para tiempo-real, ya que es una tecnología para sistemas genéricos. Los ensamblados de .NET presentan las mismas carencias para tiempo-real que pueda presentar otros modelos orientados a componentes [9].
- Utilizar bibliotecas de lenguaje *C* o *C#* para acceder al *Process Control Block* (*PCB*), una estructura de datos en la que el sistema operativo registra el tiempo efectivo de ejecución de cada proceso. Sin embargo, investigando en tesis, artículos y libros, se determinó que esta alternativa, aunque interesante, no era tan sencilla de implementar. La razón consiste en que los lenguajes de programación de alto nivel se encuentran hasta arriba en la estructura del *Framework* de .Net y todo el código pasaba a convertirse en *Common Intermediate Lenguage* o mejor conocido como *MSIL* (*Microsoft Intermediate Lenguage*) por el CLR [10]. El acceso al PCB (*Process Control Block*) es altamente restringido por el S.O. dado que organiza en un conjunto de campos en los que se almacena información de diversos tipos: información de identificación, información de estado de la CPU, información de control del proceso e información de uso de recursos.
- Controlar ambiente de experimentación. La idea consiste en ejecutar la aplicación en la computadora,

quitando todos los programas innecesarios de tal forma que sean solo procesos del S.O., dándonos unos resultados un poco más confiables.

Para ejecutar los experimentos se utilizó el último enfoque. Los resultados obtenidos se muestran en la Tabla II. Puede observarse que los resultados muestran un mejor desempeño con Mono. A pesar de que la diferencia no es tan grande, en la medida que el volumen de operaciones se incremente, ésta diferencia puede ser más significativa.

V. CONCLUSIONES Y TRABAJO FUTURO

Para poder desarrollar aplicaciones corporativas de manera eficiente y ágil, que sean capaces de ejecutarse en plataformas heterogéneas y conectarse a sistemas desarrollados con herramientas diversas, es necesaria la utilización de marco de referencia robusto. En este contexto, .Net de Microsoft se ha consolidado como una de las herramientas más utilizadas. Debido a que utiliza software propietario, recientemente ha proliferado el uso de Mono, que es una plataforma compatible con el marco de referencia de .Net y que permite la utilización de software libre.

En este artículo se presentó un análisis comparativo de .Net y Mono, y se estudiaron sus principales características. Después de analizar los resultados, se puede concluir que la plataforma con mejor desempeño fue Mono. A pesar de que la diferencia en desempeño no es muy grande, cuando el volumen de transacciones es elevado, ésta diferencia puede ser importante. Por otra parte, se mostró que a pesar de que .Net de Microsoft tiene más soporte que Mono y tiene más tiempo en el mercado, Mono y su entorno de desarrollo son herramientas maduras que cumplen con el marco de referencia .Net. El trabajo a futuro que se tiene planeado es realizar aplicaciones móviles bajo el marco de referencia de .Net integrando servicios *web* y arquitectura orientada a servicios además de hacer la comparación con la plataforma Mono.

REFERENCIAS

[1] F. Ramirez, *Aprenda Practicando ASP.NET usando Visual Studio 2012*. Alfaomega, 2012.
[2] D. S. Platt, *Introducing Microsoft .Net, Second Edition*. Microsoft Press, 2001.
[3] E. INTERNATIONAL, “Ecma-335: Common language infrastructur (cli),” 2010.
[4] *www.microsoft.com/visualstudio*.
[5] I. Griffiths, *Programing Csharp 5.0*. O ’Reilly Media, 2012.
[6] E. INTERNATIONAL, “Ecma-334: Csharp language specification,” 2006.
[7] *www.monodevelop.com*.
[8] D. B. Stewart, “Measuring execution time and real-time performance,” in *Proceedings of the Embedded Systems Conference*, (Boston, Ma), pp. 1—15, 2006.
[9] D. G. Marquez, *Componentes Software para Sistemas Distribuidos de Tiempo Real*. PhD thesis, Universidad de Malaga, 2005.
[10] M. Mamone, *Practical Mono*. Apress, 2006.