Interoperabilidad entre lenguajes de programación

Interoperability between programming languages

Severino Feliciano Morales*
José Luis Hernández Hernández**
René Edmundo Cuevas Valencia***

Fecha de recepción: 15 de abril de 2011
Fecha de aceptación: 16 de junio del 2011

Resumen

Este artículo tiene como finalidad explicar de qué manera se da la interoperabilidad entre los lenguajes de programación desarrollados o modificados para la tecnología .NET. Esta nueva forma de desarrollo de software, conocida también como programación de lenguaje mixto, se trata de que el código generado por un lenguaje pueda funcionar fácilmente con el código generado por otro lenguaje. Este enfoque de desarrollo de programas hace que se faciliten las cosas para crear grandes sistemas distribuidos de software, para la programación orientada a componentes, ya que si un componente puede ser utilizado por la mayor variedad posible de lenguajes de computación y por el mayor número de entornos operativos, se considera, además de eficiente muy valioso (Schildt, 2003). La interoperabilidad entre lenguajes es la posibilidad de que el código interactúe con código escrito en un lenguaje de programación diferente. La interoperabilidad entre lenguajes puede ayudar a maximizar la reutilización de código y, por tanto, puede mejorar la eficacia del proceso de programación.

* Unidad Académica de Ingeniería de la Universidad Autónoma de Guerrero Av. Lázaro Cárdenas s/n, Ciudad Universitaria. Chilpancingo Guerrero México. Teléfono: 01 (747) 47 2-79-43. Correo electrónico: sevefelici@hotmail.com

** Unidad Académica de Ingeniería de la Universidad Autónoma de Guerrero Av. Lázaro Cárdenas s/n, Ciudad Universitaria. Chilpancingo Guerrero México. Teléfono: 01 (747) 47 2-79-43. Correo electrónico: tec_jlhh05@yahoo.com.mx

*** Unidad Académica de Ingeniería de la Universidad Autónoma de Guerrero Av. Lázaro Cárdenas s/n, Ciudad Universitaria. Chilpancingo Guerrero México. Teléfono: 01 (747) 47 2-79-43. Correo electrónico: reneecuevas@hotmail.com

Palabras clave: interoperabilidad, CLR, CIL, MSIL, portabilidad, multilenguaje, .NET, POO, paradigmas, multiplataforma.

Abstract

This article aims to explain how there is interoperability betweenprogramming languages developed or modified. NET technology. This new form of software development, also known as mixed-language programming, this is the code generated by a language can easily run the code generated by anotherlanguage. This approach to software development,itmakesthingseasiertobuildlargedistributedsystems software forcomponent-oriented programming, because if a component canbe used by the widest variety of computer languages and as many of operating environments, are considered very valuableas well as efficient (Schildt, 2003) [1]. Language interoperability is the ability to interact with sourcecode written in different programming language. Language interoperability can help maximize code reuse and, therefore, can improve the effectiveness of the programming process.

Key words: : Iinteroperabilidad, CLR, CIL, MSIL, portability, multi-language NET, OOP paradigm, multiplatform

Introducción

Para entender claramente cómo es que ahora se plantean diferentes formas de solucionar problemas de automatización por medio del proceso de desarrollo de software, se deben identificar las diferentes formas de pensar o paradigmas y que se pueda comprender cómo ha evolucionado de manera exorbitante todo el ámbito de la programación con el paso del tiempo.

Principales paradigmas de programación

En la programación imperativa, se describe paso a paso un conjunto de instrucciones que se deben ejecutar para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema. En los declarativos, las sentencias que se utilizan describen el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información, a partir de la descripción realizada.

Los imperativos tienden a enfatizar la serie de medidas adoptadas por un programa para llevar a cabo una acción, mientras que los programas funcionales tienden a enfatizar la composición y el arreglo de funciones, a menudo sin especificar explícitamente los pasos.

Los declarativos y las sentencias que se utilizan describen el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada (León Rivera, s.f.), como se muestra en la tabla 1.

Tabla 1. Comparación de paradigmas

Aspectos	Imperativo	Declarativo
Implementación de la solución.	Especifica el cómo.	especifica el qué.
Orden de ejecución.	Se especifica el algoritmo escrito por el programador.	no se especifica.
Responsable del control de ejecución del programa.	Comandos y estructuras de datos.	algoritmo de resolución del software.
Elementos primarios.	Destructiva con efectos laterales.	relaciones, reglas o funciones.
Asignación de variables.	Sí usa variables y estados.	No destructiva, sin efectos laterales.
Noción clásica de variables y estados		Evade estados y variables.

Evolución de la programación

Durante años, los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser utilizados por otras personas se creó la POO. Esta es una serie de normas de realizar las cosas de manera que otras personas puedan utilizarlas y adelantar su trabajo, de manera que consigamos que el código se pueda reutilizar. Los conceptos de la programación orientada a objetos tienen origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard del Centro de Cómputo Noruego en Oslo. Según se informa, la historia es que trabajaban en simulaciones de naves y fueron confundidos por la explosión combinatoria de cómo las diversas cualidades de diversas naves podían afectar unas a las otras. La idea ocurrió para agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamiento. Fueron refinados más tarde en Smalltalk, que fue desarrollado en Simula en Xerox PARC. pero diseñado para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar "en marcha" en lugar de tener un sistema basado en programas estáticos.

La programación orientada a objetos tomó posición como la metodología de programación dominante a mediados de la década de los ochenta, en gran parte, debido a la influencia de C++, una extensión del lenguaje de programación C. Su dominación fue consolidada gracias al auge de las interfaces gráficas de usuario, para los cuales la programación orientada a objetos está particularmente bien adaptada.

Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo Ada, BASIC, Lisp, Pascal, entre otros. La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas, a menudo, condujo problemas de compatibilidad y en la capacidad de mantenimiento del código. Por su parte, los lenguajes orientados

a objetos "puros" carecían de las características de las cuales muchos programadores habían venido a depender. Para saltar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos, pero permitiendo algunas características imperativas de maneras "seguras".

El Eiffel, de Bertrand Meyer, fue un temprano y moderadamente acertado lenguaje con esos objetivos, pero ahora ha sido esencialmente reemplazado por Java, en gran parte, debido a la aparición de Internet y a la implementación de la máquina virtual de Java en la mayoría de navegadores. PHP, en su versión 5, se ha modificado, soporta una orientación completa a objetos, cumpliendo con todas las características propias de esta. A lo largo de los años, los lenguajes de programación orientados a objetos han evolucionado de manera exorbitante.

En la actualidad, el paradigma de orientación a objetos es, sin lugar a dudas, el más utilizado por las empresas de todo el mundo a la hora de encarar desarrollos de aplicaciones de software, ya que permite representar de manera relativamente simple modelos y realidades muy complejas y esto hace que el software sea más fácil de programar, comprender y mantener. Por otra parte, luego de más de 20 años de investigación y de desarrollo sobre orientación a objetos, pareciera ser que la industria se ha dado cuenta de que el paradigma está lo suficientemente maduro como para dar soporte a las aplicaciones más importantes del mundo actual.

El primer paso para llegar a la programación orientada a objetos (POO: *Object Oriented Programming*) es la programación modular, que busca aplicar sanos principios de abstracción para dividir un problema de programación muy complejo en módulos, o partes, manejables. La POO, simplemente, agrega un nuevo tipo de módulo, el objeto, que corresponde a las variables que siempre es necesario manipular en los programas. Sin datos, un programa no puede realizar trabajo útil alguno, por lo que es natural que las variables tengan el mismo rango que los otros módulos que conforman un programa. El trabajo que un programa realiza es útil porque modifica sus datos.

La programación por objetos busca la reutilización de componentes de programas. Tiene fuertes bases en el uso de bibliotecas de programas, que ya se han utilizado por casi treinta años. La idea original de usar POO nace con el lenguaje *Simula,* a finales de la década de los sesenta. Se ha dicho que POO es un paradigma de programación porque, al usar POO, el programador trata de enfocar la programación, no solo desde el punto de vista de los procesos algorítmicos, sino que también toma muy en cuenta los datos. La idea general es que cada dato (objeto) es un componente de programación.

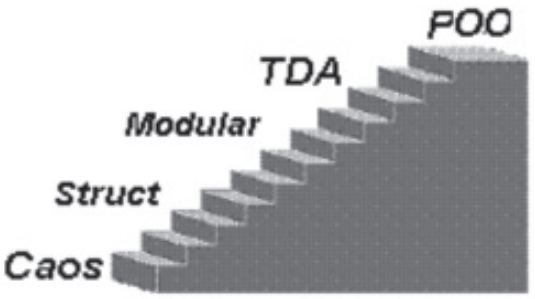
Con la introducción de sistemas operativos gráficos, como Windows, ha surgido un nuevo concepto de programación. Los programadores ahora diseñan aplicaciones a base de unir diferentes piezas de código ya escrito y probado con anterioridad, cada una de estas piezas se llama "objeto". Los objetos pueden tener propiedades, como forma, tamaño, color, y tipo de datos. Un ejemplo podría ser un cuadro en la pantalla conteniendo una cuenta de dinero, se podría cambiar el tamaño de este cuadro, cambiar el color, cambiar la forma en la que se muestra la figura de la cuenta de dinero. Entonces se podría realizar operaciones en esa cuenta, controlando los eventos que se hicieran sobre ella, como una pulsación del ratón que podría disparar una determinada operación, o mover la caja sobre la pantalla. Gracias al entorno del sis-

tema operativo subyacente tras ello, se puede hacer todo esto poniendo propiedades en lugar de escribir todo un código. Y no solo esto, sino que también se puede crear una aplicación con una base de datos subyacente tras ella, utilizando todas las plantillas preprogramadas sobre la estructura de la base de datos. En los días del antiguo lenguaje ensamblador, este sistema hubiera sido extraordinario, si no imposible, de llevarlo a la práctica, además el tamaño y la complejidad serían horrendos.

Pero, mucha de esa complejidad está ya incluida en el sistema operativo de las computadoras que ejecutan estos programas "orientados a objetos". Lo bueno de esto es que muchas de las funciones que antes llevaban meses para programarlas, hoy en día, se pueden hacer fácilmente en unos días, incluso en cuestión de horas. Además, el costo económico de los sistemas operativos de la computadora se ha revolucionado, debido a que los costos del hardware caen en picado.
La orientación a objetos (OO), que inicialmente fue un conjunto de técnicas de programación soportadas en el uso de lenguajes especiales (orientados a objetos), ha ido poco a poco más allá de la propia programación hasta convertirse en una metodología genérica y de gran potencia para construir modelos de sistemas, que puede ser aplicada en todas las fases del desarrollo de aplicaciones: análisis, diseño, programación y mantenimiento.
En relación con otras metodologías, tiene la ventaja de ser más natural –más próxima a la forma de pensar y hablar de las personas– e integrar los principios generales de la ingeniería del software en un paradigma coherente (el concepto de "objeto").

En la actualidad, los conceptos fundamentales de objetos están bien asentados. Un modelo de objetos es un conjunto de entidades (denominadas objetos) que colaboran entre ellos para desempeñar una serie de servicios. Esos servicios se solicitan por medio del intercambio de mensajes. Todos los objetos del modelo pertenecen a algún tipo (clase).
Además, los objetos pueden ser organizados en jerarquías, de forma unos objetos pueden heredar datos y algoritmos de otros objetos. Con esto lo que se consigue es que la organización de un programa orientado a objetos sea más modular y rica que la organización de un programa estructurado, con lo que la arquitectura de los programas complejos puede ser acomodada a cambios más fácilmente (Carrera Díaz, s.f.), como se muestra en la figura 1.

Figura 1. Eficiencia de la POO



Actualmente, la programación va más allá que la OO, asimismo, se piensa en grandes sistemas distribuidos de software, programación orientada a componentes y servicios web, en los que se piensa en trabajar con elementos de software desarrollados cada uno en diferentes lenguajes y mediante compilación hibrida.

Dado que los desarrolladores utilizan una gran variedad de herramientas y de tecnologías, cada una de las cuales podría admitir distintos tipos y características, desde tiempo atrás ha sido complicado garantizar la interoperabilidad entre lenguajes. No obstante, los compiladores y las herramientas de lenguaje dirigidos a *Common Language Runtime*

(CLR) se benefician de la compatibilidad que integra el motor en tiempo de ejecución para la interoperabilidad entre lenguajes.

El código administrado se beneficia en que el motor en tiempo de ejecución admita la interoperabilidad entre lenguajes de las siguientes maneras:

Los tipos pueden heredar la implementación de otros tipos, pasar objetos a los métodos de otro tipo y llamar a métodos definidos para otros tipos, sea cual sea el lenguaje en que se implementen los tipos.

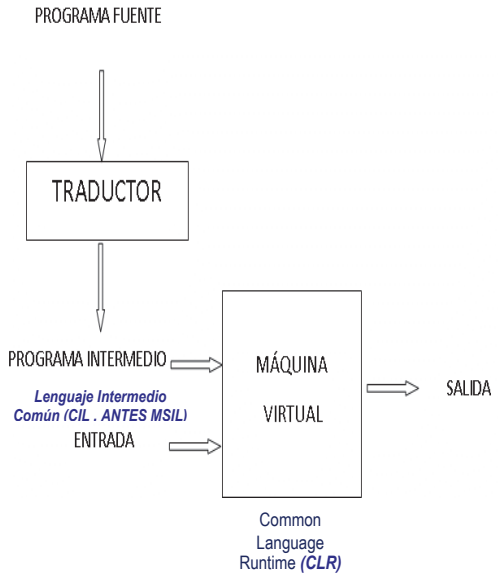
Los depuradores, generadores de perfiles u otras herramientas deben reconocer un solo entorno, es decir, MSIL (Microsoft *intermediate language,* Lenguaje intermedio de Microsoft) y los metadatos de *common language runtime*, para poder ser compatibles con cualquier lenguaje de programación dirigido al motor en tiempo de ejecución.

El control de excepciones es coherente entre todos los lenguajes. El código puede producir una excepción en un lenguaje y esa excepción puede ser recibida y reconocida por un objeto escrito en otro lenguaje (MSDN, 2011)

Compilación híbrida

Ahora bien, para trabajar de esta maneara, el código fuente del lenguaje que se maneja frecuentemente no se traduce directamente al código ejecutable, sino que para que garantice la portabilidad y la posible integración con componentes realizados en otros lenguajes, se tiene que implementar lo que comúnmente se llama como compilación hibrida, es decir, primero se traduce a un código intermedio antes de que genere el ejecutable (Aho et ál., s.f.), como se muestra en la figura 2.

Figura 2. Compilación híbrida



Fuente: elaboración propia.

El Framework .NET de Microsoft es independiente del lenguaje utilizado, lo que significa que los que escojamos para nuestras aplicaciones deben ser traducidos a un lenguaje binario comprensible por la plataforma. Por esta razón, los desarrolladores que utilizan los servicios de esta plataforma pueden emplear varios lenguajes de alto nivel y hacerlos operar entre sí (Windu, 2011).

Es un modelo de programación consistente y sencillo, completamente orientado a objetos, en el que se elimina el temido problema de compatibilidad entre DLL, ejecución multiplataforma, ejecución multilenguaje, hasta el punto 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.

Las herramientas más recientes de Microsoft, que trabajan con este tipo de compilación, son lenguajes desarrollados y modifica-

dos para la tecnología .NET y forman parte de la estrategia para integrar Internet y Web, en las aplicaciones de computadora. Esta tecnología les proporciona a los desarrolladores las herramientas que necesitan para crear y ejecutar aplicaciones de computadora que pueden ejecutarse en computadoras distribuidas por medio de Internet. Los tres principales lenguajes de programación son Visual Basic .Net (basado en el lenguaje Basic original), Visual C++ (basado en C++) y C# (basado en C++ y Java) (León Rivera, s.f.).

El lenguaje C# es el único que fue diseñado especialmente para tecnología .Net, los demás fueron adaptados o modificados, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes. Por esta razón, Microsoft suele referirse a C# como el lenguaje nativo de .NET, y de hecho, gran parte de la librería de clases base de .NET ha sido escrito en este lenguaje.

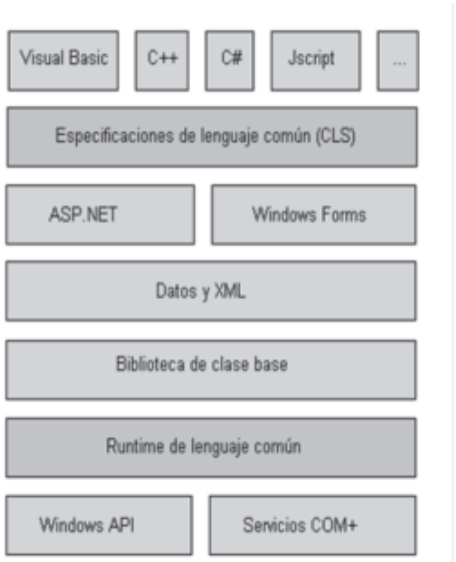
C# es un lenguaje orientado a objetos sencillo, moderno, amigable, intuitivo y fácilmente legible que ha sido diseñado por Microsoft con el ambicioso objetivo de recoger las mejores características de muchos otros lenguajes, fundamentalmente Visual Basic, Java y C++, y combinarlas en uno solo en el que se unan la alta productividad y facilidad de aprendizaje de Visual Basic con la potencia de C++ (González Seco, s.f.). Microsoft ha creado compiladores para Visual Basic, C++ y C#, aunque también existen compiladores para esta plataforma que no han sido creados por Microsoft.

Tecnología .NET

Es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con la finalidad de mejorar sus sistemas operativos, mejorar su modelo de componentes, 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 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 como .NET Framework. El corazón de la plataforma .NET es el CLR (*common language runtime*), que es una aplicación similar a una máquina virtual que se 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 (Fundación Josep Carreras, 2011). La figura 3 muestra en forma gráfica la representación de .NET Framework.

Figura 3. Entorno de NET Framework



Fuente: Adrformacion (2011).

Un framework es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar. Asimismo, es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado

Common language runtime o CLR (entorno en tiempo de ejecución de lenguaje común)

Es un entorno de ejecución para los códigos de los programas que corren sobre la plataforma Microsoft .NET. El CLR es el encargado de compilar una forma de código intermedio llamada *common intermediate language* –CIL, anteriormente conocido como MSIL, por Microsoft Intermediate Language–, al código de maquina nativo, mediante un compilador en tiempo de ejecución. No debe confundirse el CLR con una máquina virtual, ya que una vez que el código esta compilado, corre nativamente sin intervención de una capa de abstracción sobre el hardware subyacente (Wikipedia).

Los desarrolladores que usan CLR escriben el código fuente en un lenguaje compatible con .NET, como C# o Visual Basic .NET. En tiempo de compilación, un compilador .NET convierte el código a CIL. En tiempo de ejecución, el compilador en tiempo de ejecución del CLR convierte el código CIL en código nativo para el sistema operativo. Alternativamente, el código CIL es compilado a código nativo en un proceso separado anterior a la ejecución. Esto acelera las posteriores ejecuciones del software, debido a que la compilación de MSIL a nativo ya no es necesaria. A pesar de que algunas implementaciones del *common language infrastructure* se ejecutan en sistemas operativos que no sean Windows, el CLR se ejecuta solo en Microsoft Windows. El *runtime* del lenguaje común es la primera capa que pertenece a .NET Framework. Esta capa es la responsable de los servicios básicos de .NET, como la administración de memoria, la recolección de los elementos no utilizados, el control estructurado de excepciones y del subprocesamiento múltiple. Si .NET se transporta a otras arquitecturas que no estén basadas en Windows el primer paso sería escribir un *runtime* del lenguaje para el nuevo equipo. El CLR tiene las siguientes características:

- Administra el código en tiempo de ejecución: carga en memoria, liberación de memoria.
- Gestiona la seguridad del código ejecutado.
- Abre posibilidades a otros fabricantes para incorporar sus lenguajes.
- Facilita la distribución e instalación de aplicaciones.
- Elimina los temibles conflictos de DLL y versiones de ellas.
- Es la interfaz entre nuestro código y el sistema operativo (Adrformacion, 2011).

Common intermediate language (CIL)

Pronunciado “sil” o “kil”, anteriormente llamado Microsoft Intermediate Language o MSIL, es el lenguaje de programación de más bajo nivel en el *common language infrastructure* y en el .NET Framework. Los lenguajes del .NET Framework compilan a CIL. CIL es un lenguaje ensamblador orientado a objetos y está basado en pilas y es ejecutado por el CLR. Los lenguajes .NET principales son C#, Visual Basic .NET, y J# (Wikiperdia).

Common language infrastructure (CLI)

La infraestructura de lenguaje común, es una especificación estandarizada que describe un

entorno virtual para la ejecución de aplicaciones, cuya principal característica es la de permitir que aplicaciones escritas en distintos lenguajes de alto nivel puedan luego ejecutarse en múltiples plataformas tanto de hardware como de software sin necesidad de reescribir o recompilar su código fuente (Wikipedia).

Proceso de ejecución de una aplicación

El proceso de ejecución administrado incluye los siguientes pasos:

1. La elección de un compilador. Para obtener los beneficios proporcionados por *common language runtime*, debe utilizar compiladores de lenguaje que se dirigen el tiempo de ejecución.
2. Compilar el código en MSIL. La compilación traduce el código fuente en lenguaje intermedio de Microsoft (MSIL) y genera los metadatos necesarios.
3. Compilar MSIL a código nativo. En tiempo de ejecución, un "justo a tiempo" (JIT) compilador traduce el código MSIL a código nativo. En esta compilación, el código debe pasar un proceso de verificación que examina el MSIL y los metadatos para determinar si el código puede determinar qué tipo de seguro.
4. Ejecución de código. El *common language runtime* proporciona la infraestructura que permite la ejecución tenga lugar y los servicios que se pueden utilizar durante la ejecución (MSDN, 2011).

Conclusiones

Es evidente, que la transformación de las herramientas y las formas de desarrollo de software evolucionan de manera vertiginosa con el paso de los días. Por lo que se hace importante hacer un análisis profundo, antes de decidir qué herramientas o qué lenguajes de programación utilizar para el desarrollo de software.

Es indudable que existen lenguajes potentes y que crean aplicaciones robustas y, por tanto, las personas que trabajan con algún lenguaje, como Java, siempre van afirmar que no hay otro lenguaje mejor; sin embargo, a pesar de crear programas a gran escala y aparatos domésticos, no es totalmente compatible con Windows que sigue siendo el sistema operativo más popular del mundo y no facilita la interoperabilidad.

En consecuencia, la neutralidad del lenguaje ha incrementado mucho el atractivo de la tecnología .NET (pricipalmente c#), en especial, cuando se considera el coste que es necesario afrontar para migrar desde otras plataformas. De hecho, la migración desde J2EE es prácticamente inmediata, ya que una aplicación Java puede convertirse sin mucho esfuerzo en un programa .NET utilizando compiladores cruzados. Es más, cualquier grupo de desarrolladores Java puede adoptar C# como lenguaje sin graves consecuencias.

C# es el lenguaje abanderado por Microsoft como estandarte para los programadores .NET y a la par, la herramienta con la que persigue la captación de desarrolladores de otras plataformas. No en vano, para atraer programadores procedentes del entorno Java, Microsoft creó una iniciativa denominada JUMP (Java Users Migration Path), que significa algo así como "el camino que debe seguir un desarrollador Java para programar en .NET". En cualquier caso, Java y C# son dos lenguajes similares, sobre todo, en el nivel semántico, una capacidad que permite la existencia de aplicaciones que facilitan la conversión entre ambos lenguajes.

Referencias

Adrformacion (2011). Recuperado el 15 de mayo del 2011 de: http://www.adrformacion.com/curso/puntonet/leccion1/tecnologia_punto_net.htm.

Aho, A. et ál. (comp.). (s.f.). *Principios, técnicas y herramientas*. Pearson Addison Wesley.

Carrera Díaz, V. (s.f.). *Características de la POO*. Unidad Académica de Ingeniería de la Universidad Autónoma de Guererero.

Deitel, P.J. y Deitel, H.M. (s.f.). *Java. Cómo programar* (7ª ed.). Pearson Prentice Hall.

Fundación Josep Carreras (2011). *¿Qué es .NET?* Recuperado el 14 de mayo del 2011 de: http://globaliza.blogia.com/temas/tecnologia.net.php.

González Seco, J.A. (s.f). *C# El nuevo lenguaje de Internet*. Recuperado el 6 de mayo del 2011 de: http://mygnet.com.

León Rivera, S. (s.f.). *Análisis comparativo de los principales paradigmas de programación*. Unidad Académica de Ingeniería de la Universidad Autónoma de Guererero.

MSDN (2011). *Interoperabilidad entre lenguajes*. Recuperado el 15 de mayo del 2011 de: http://msdn.microsoft.com/es-es/library/a2c7tshk.aspx.

Schildt, H. (2003). *C#. Manual de referencia*. Mc Graw Hill.

Wikipedia. Recuperado el 3 de mayo del 2011 de: http://es.wikipedia.org/wiki/.

Windu, M. (2011). Recuperado el 16 de mayo del 2011 de: http://www.portalhacker.net/index.php/topic,63240.0.html.