Desarrollo de aplicaciones, utilizando software libre y propietario

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

Álvarez Hilario Valentín
Unidad Académica de Ingeniería de la Universidad Autónoma de Guerrero Av. Lázaro Cárdenas s/n, Ciudad Universitaria. Chilpancingo Guerrero México
01 (747) 47 2-79-43
valentin_ah@yahoo.com

Hernández Hernández José Luis
Unidad Académica de Ingeniería de la Universidad Autónoma de Guerrero Av. Lázaro Cárdenas s/n, Ciudad Universitaria. Chilpancingo Guerrero México
01 (747) 47 2-79-43
tec_jlhh@yahoo.com.mx

Fecha de recepción: 17 de enero de 2012
Fecha de aceptación: 28 de febrero de 2012

Resumen

Este artículo tiene como finalidad describir la forma de cómo se implementa la programación de lenguaje mixto, la cual permite desarrollar aplicaciones específicamente en lenguajes desarrollados o modificados para la tecnología .NET. No obstante, se hace énfasis en cómo utilizar el código generado en java por otros lenguajes de programación. Esta forma de desarrollo de software, conocida también como interoperabilidad entre lenguajes de programación diferentes, es la capacidad de dos o más componentes desarrollados en diferentes lenguajes, para intercambiar información

y utilizar la información intercambiada [14]; es la posibilidad de que cierto código interactúe con código escrito en un lenguaje de programación diferente. 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 y 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 [1].

La interoperabilidad entre lenguajes puede ayudar a maximizar la reutilización de código y por tanto, puede mejorar la eficacia del proceso de programación [1].

Para que se pueda desarrollar una aplicación con estas características, se hace necesario entender la compilación híbrida, que combina el proceso de compilación con interpretación [4], que naturalmente se aborda también en el artículo.

Así mismo, se explica como se ha estado tratando este tipo de desarrollo en proyectos de software libre como en software propietario.

Palabras Clave: Programación de lenguaje mixto, Interoperabilidad, CLR, CIL, MSIL, portabilidad, multilenguaje, .NET, POO, paradigmas, multiplataforma, mono, c#, java, máquina virtual, BCL, software libre, software propietario, compilación híbrida, IKVM.NET, DLL.

Abstract

This article aims to describe how to how to implement the mixed-language programming, which allows development of applications specifically developed or modified language. NET technology. However, the emphasis is on how to use the java code generated by other programming languages. This form of software development, also known as interoperability between different programming languages is the ability of two or more components developed in different languages, to exchange information and use the information exchanged [14], is the possibility that some code to interact with code written in different programming language. This is the code generated by a language can easily work with the code generated by another language. This approach to program development, does that make things easier to build large distributed systems and software for component-oriented programming, because if a component can be used by the widest possible range of computer languages and as many of operating environments, is considered, as well as efficient, very valuable [1].

Language interoperability can help maximize code reuse and thus may improve the effectiveness of the programming process [1].

To be able to develop an application with these characteristics, it is necessary to understand the build hybrid, combining the process of compilation with interpretation [4], which naturally is also addressed in the article.

It also explains how people have been trying this type of development in open source projects such as proprietary software.

Keybwords: Mixed-language programming, Interoperability, CLR, CIL, MSIL, portability, multi-language. NET OOP paradigm, multimono, c #, java virtual machine, BCL, free software, proprietary software, build hybrid IKVM.NET, DLL.

1. Introducción

La programación es el proceso de diseñar, escribir, depurar y mantener el código fuente de programas computacionales. El código fuente es escrito en un lenguaje de programación y su propósito es crear programas que exhiban cierto comportamiento deseado. El proceso de escribir código requiere frecuentemente conocimientos en varias áreas distintas, además del dominio del lenguaje a utilizar, algoritmos especializados y lógica formal.

Programar no involucra necesariamente otras tareas tales como el análisis y diseño de la aplicación (pero sí el diseño del código), aunque sí suelen estar fusionadas con la implementación de sistemas de información.

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 se pueda comprender como ha evolucionado de manera exorbitante todo el ámbito de la programación con el paso del tiempo.

1.1 Paradigmas de programación

En la programación imperativa se describe paso a paso un conjunto de instrucciones que deben ejecutarse 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.

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 las sentencias que se utilizan lo que hacen es describir 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 [3].

1.2 Programación Orientada a Objetos (POO)

La POO, que fue la revolución en la forma de desarrollar software, es en la actualidad la más usada en los lenguajes de programación, por lo que este paradigma se merece analizar por separado.

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; que consta de una serie de normas para realizar las cosas de manera que otras personas las utilicen, mejorando su productividad, a ésta acción se le conoce como reutilización de código.

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 se diseñó 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 los años 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. La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas, condujo a menudo a problemas de compatibilidad y en la capacidad de mantenimiento del código.

Los lenguajes orientados a objetos "puros", por su parte, 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 prácticamente remplazado por Java, en gran parte debido a la aparición de Internet y a la implementación de la máquina virtual de Java.

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 desarrollo sobre Orientación a Objetos pareciera ser que la industria se ha dado cuenta 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. Un objeto, es la instancia de una clase, el cual al ser creado, automáticamente posee todas las propiedades de la clase a la que pertenece.

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. Es un paradigma de programación, porque el programador trata de enfocar la programación, no sólo 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.

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, tales como forma, tamaño, color y tipo de datos. Un ejemplo podría ser un cuadro en la pantalla conteniendo una cuenta bancaria por ejemplo, se podría cambiar el tamaño de este cuadro, cambiar el color, cambiar la forma en la que se muestra la figura de la cuenta. Entonces se pueden realizar operaciones en esa cuenta, controlando los eventos que se hacen sobre ella, tales como una pulsación del ratón que podría disparar una determinada operación, o mover la caja sobre la pantalla. Gracias al entorno del sistema operativo subyacente a la aplicación, se puede hacer todo esto colocando propiedades en lugar de escribir todo un código.

Pero mucha de esa complejidad está ya incluida en el sistema operativo de las computadoras que ejecutan estos programas "orientados a objetos". Lo interesante de esto es que muchas de las funciones que antes llevaban meses para programarlas, hoy 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 han disminuido.

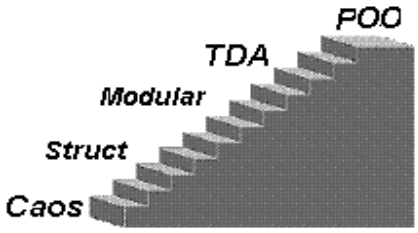
La Orientación a Objetos (OO), que inicialmente fue un conjunto de técnicas de programación soportadas en el uso de lenguajes especiales (orientados a objetos), ha ido poco a poco más allá de la propia programación hasta convertirse en una metodología genérica y de gran potencia para construir modelos de sistemas, que puede ser aplicada en todas las fases del desarrollo de aplicaciones: análisis, diseño, programación y mantenimiento.

Frente a 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").

Los conceptos fundamentales de Objetos están en la actualidad 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 que unos objetos pueden heredar datos y métodos 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 [10], como se muestra en la Figura 1.

Figura 1. Jerarquía de la POO.



1.3 Interoperabilidad entre lenguajes de programación

Como ya se había comentado, la interoperabilidad, se trata de que el código generado por un lenguaje pueda funcionar fácilmente con el código generado por otro lenguaje.

Este tipo programación va más allá que la OO, sino que se piensa en grandes sistemas distribuidos de software, programación orientada a componentes y servicios web, donde se piensa en trabajar con elementos de software desarrollados cada uno en diferentes lenguajes y a través de compilación hibrida.

Dado que los desarrolladores utilizan una gran variedad de herramientas y 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, que es el código desarrollado para .Net, se beneficia debido a que el motor en tiempo de ejecución admite la interoperabilidad entre lenguajes y brinda las bondades siguientes:

- 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, CIL (Common Intermediate Language), Lenguaje Intermedio Común 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 [6][13].

2. Compilación hibrida

Para que se pueda desarrollar una aplicación con estas características, se hace necesario entender la compilación híbrida, que combina el proceso de compilación con interpretación.

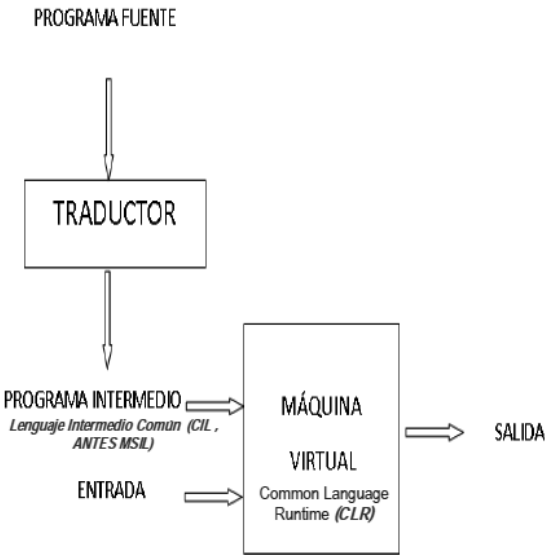
El proceso de compilación de Java no genera un código ejecutable, sino un bytecode. El bytecode es un conjunto de instrucciones altamente optimizado diseñado para ser interpretado por una máquina virtual la cuál es llamada la máquina virtual de java (JVM) [2].

Traducir un programa Java en bytecode hace posible que la ejecución del código en plataformas distintas sea más fácil y la razón es que cada plataforma tiene su máquina virtual de Java. Las JVM difieren entre plataformas, pero todas entienden el mismo bytecode. Esto significa que el archivo .java, que es archivo fuente, se compila y se traduce a código de bytes, que es el archivo .class y este se interpreta por la JVM y hasta entonces, es ejecutado [1][2].

Ahora bien, para trabajar con código administrado, el código fuente del lenguaje que se maneja frecuentemente no se traduce directamente a código ejecutable, sino que para que garantice la portabilidad y la posible integración con componentes realizados en otros lenguajes, El Common Language Runtime (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.

El CLR es muy diferente a 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 [11]. En cambio para el proceso de ejecución de java, se tiene que implementar lo que comúnmente se llama como compilación hibrida, es decir, primero se traduce a un código intermedio y después se interpreta [4], tal como se muestra en la figura 2.

Figura 2. Compilación hibrida [13].



Un Framework .NET como el 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 [8].

Es un modelo de programación consistente y sencillo, completamente orientado a objetos, donde se elimina el temido problema de compatibilidad entre DLL's, 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 phyton.NET.

Las herramientas más recientes, que trabajan con este tipo de compilación, son lenguajes desarrollados y modificados para la tecnología .NET y forman parte de la estrategia para integrar Internet y Web, en las aplicaciones de computadora. Esta tecnología proporciona a los desarrolladores las herramientas que necesitan para crear y ejecutar aplicaciones de computadora que pueden ejecutarse en computadoras distribuidas a través de Internet.

Cabe señalar, que 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. Ha sido tanta la influencia de este tipo de programación, que hasta ya existe C# y mono basic (que es el equivalente a visual basic .Net) para software de fuente abierta [1].

C# es un lenguaje orientado a objetos sencillo, moderno, amigable, intuitivo y fácilmente legible que ha sido diseñado inicialmente 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 sólo en el que se unan la alta productividad y facilidad de aprendizaje de Visual Basic con la potencia de C++ [5].

Microsoft ha creado compiladores para Visual Basic, C++ y C#, aunque ya existen también una variedad de compiladores para esta plataforma que no han sido creados por Microsoft, como el mismo C# de software libre.

3. Microsoft .NET

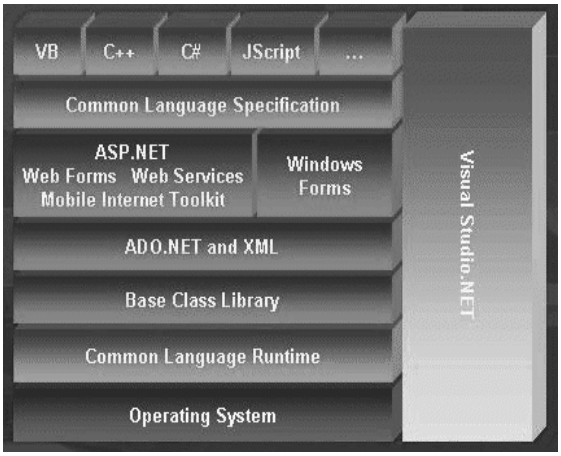
Es el conjunto de nuevas tecnologías en las que Microsoft ha trabajado los últimos años con la finalidad de mejorar sus sistemas ope-

rativos, 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. Éste 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 facilita su desarrollo y mantenimiento y favorecen su fiabilidad y seguridad [7][13].

La figura 3. Entorno de NET Framework [12].



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.

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 [12].

3.1 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. Common Language Runtime 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 [11].

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, tales 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's y versiones de ellas.
- Es la interfaz entre nuestro código y el sistema operativo [9]

3.2 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# [11][13].

3.3 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 [11][13].

4. Proceso de ejecución con Microsoft .NET

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 a 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 se puede determinar que 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 [6].

5. Implementación de la interoperabilidad con visual studio .NET

Para ejemplificar este tipo de programación, estamos utilizando un proyecto en visual studio 2010, que por medio de un web form con código en c#, se pueda utilizar clase desde un proyecto en visual basic.net, heredando elementos de una clase hecha en un proyecto en c# y posteriormente crear otro proyecto asp para que se puedan mostrar en un explorador los datos de un cliente, tomando el apellido de la clase Cliente del proyecto de visual basic y este a su vez toma el nombre de la clase Persona definida en el proyecto de c# [13], como se muestra en la figura 4, 5, 6 y 7.

Figura 4. Se define la clase Persona desde c#.

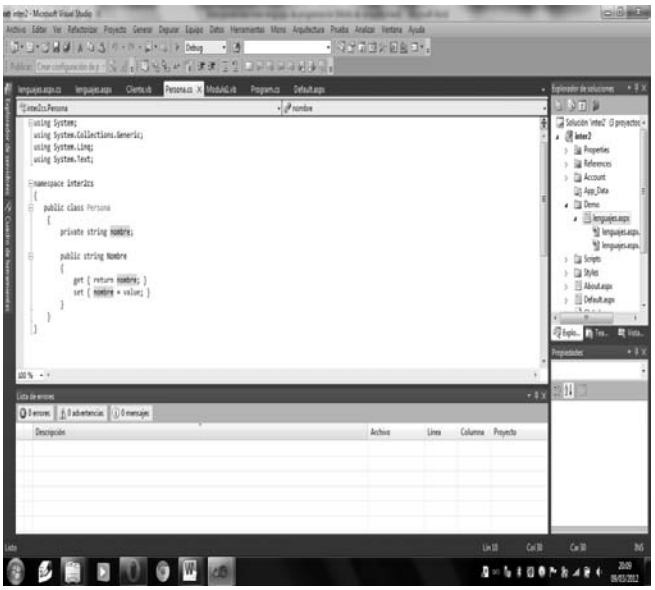


Figura 5. Se define la clase Cliente de Visual Basic, que hereda atributos desde Persona hecha en c#.

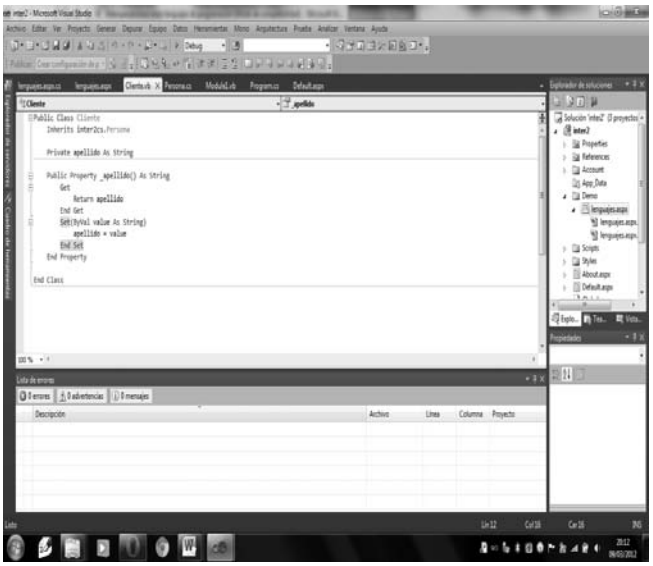


Figura 6. Se define el código de la aplicación asp para que mostrar los datos del cliente en el explorador; en este caso se usa Opera.

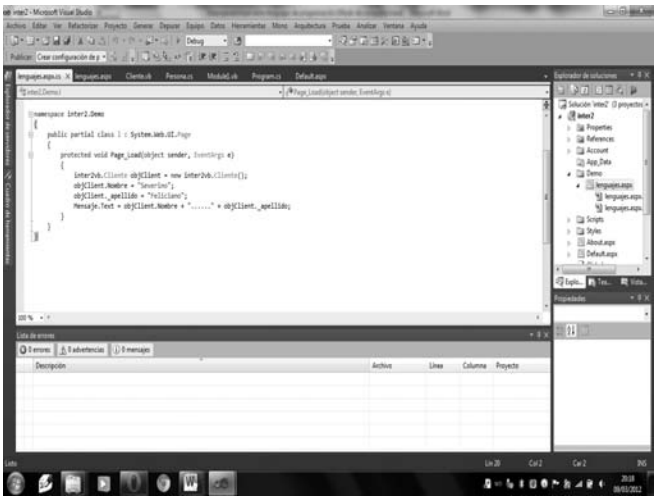
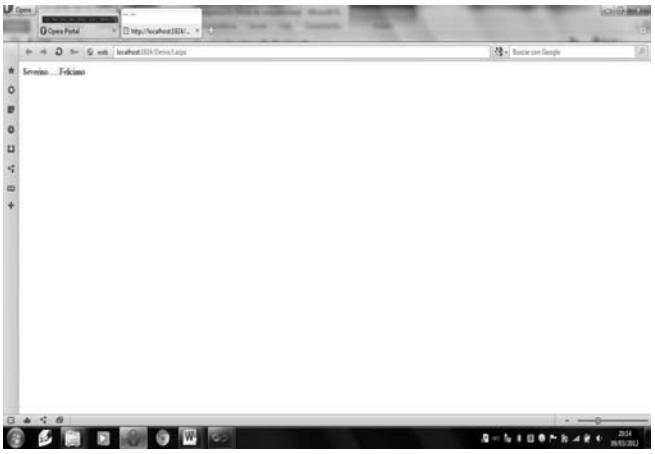


Figura 7. Se muestran los datos del cliente en el explorador.



6. Implementación de la interoperabilidad con java y c#

El compilador de Java estándar no genera un código ejecutable, sino un bytecode. El bytecode es un conjunto de instrucciones altamente optimizado diseñado para ser ejecutado por una máquina virtual la cuál es llamada la máquina virtual de java (JVM) [2].

Traducir un programa Java en bytecode hace posible que la ejecución del código en plataformas distintas sea más fácil y la razón es que cada plataforma tiene su máquina virtual de Java [1][2][4].

Sin embargo, ningún lenguaje .net, genera bytecode, sino código CIL, por lo se dificulta que el código java pueda ser usado por otros lenguajes y java no puede utilizar el código generado por otros lenguajes, pero el código java si se puede convertir en CIL para ser utilizado por otros lenguajes [14].

6.1 IKVM.NET

IKVM.NET es una implementación de Java para Mono y también para Microsoft .NET Framework., este software es libre e incluye los siguientes componentes:

- Una Máquina Virtual de Java implementada en .NET.
- Una implementación de biblioteca de clases Java en .NET.
- Herramientas de interoperabilidad entre Java y .NET.
- Con IKVM se puede ejecutar código compilado en Java en Microsoft .NET o Mono incluyendo también MonoDevelop. El código es convertido a CIL y se ejecuta.

Para ejemplificar la interoperabilidad entre Java y C#, su utiliza una clase de Java que realiza operaciones aritméticas fundamentales, como son: suma, resta, multiplicación, división y potencia. El nombre de la clase es:

OperacionesMatematicas y está incluida en el paquete: interoperabilidad.

El código de la clase (librería) de Java es el siguiente:

```
package interoperabilidad;

//Clase que realiza las operaciones básicas de aritmética
public class OperacionesMatematicas {
  //Variables de instancia de la clase
  private double suma;
  private double resta;
  private double multiplicacion;
  private double division;
  private double potencia;


  //Constructor de la clase
  public OperacionesMatematicas ()
  {
    //Inicializacion de variables de instancia
    suma=0;
    resta=0;
    multiplicacion=0;
    division=0;
    potencia=0;
  }


//Método que suma dos números y el resultado lo asigna a la  //variable de instancia suma
  public void setSuma(double a, double b)
  {
        suma= a+b;
  }

//Método que lee y devuelve el valor de la variable de instancia //suma
  public double getSuma()
  {
    return suma;
  }

  //Método que resta dos números y el resultado lo asigna a la
  //variable de instancia resta
  public void setResta(double a, double b)
  {
        resta= a-b;
  }

//Método que lee y devuelve el valor de la variable de instancia  //resta
  public double getResta()
  {
    return resta;
  }

//Método que multiplica dos números y el resultado lo asigna
//a la variable de instancia multiplicación
    public void setMultiplicacion(double a, double b)
  {
    multiplicacion= a*b;
  }

//Método que lee y devuelve el valor de la variable de instancia //multiplicación
  public double getMultiplicacion()
  {
    return multiplicacion;
  }

  //Método que divide dos números y el resultado lo asigna
  //a la variable de instancia división
  public void setDivision(double a, double b)
  {
       if(b!=0)
      division=a/b;
    else
      division=0;
  }
```

```
//Método que lee y devuelve el valor de la variable de instancia //división
  public double getDivision()
  {
    return division;
  }

//Método que obtiene la potencia de un número y el resultado   //lo asigna a la variable de instancia potencia
   public void setPotencia(double a, double b)
  {
    potencia=Math.pow(a, b);
  }

//Método que lee y devuelve el valor de la variable de instancia //potencia
  public double getPotencia()
  {
    return potencia;
  }

//fin de la clase
}
```

La clase de Java descrita anteriormente se va a utilizar en una aplicación desarrollada en C#, para esto, primero se compila la clase, puede ser con el comando javac o desde un IDE como NetBeans o eclipse, después de obtener el archivo .class, se crea el archivo .jar, este archivo es el que utiliza el comando IKVMC para obtener el archivo .dll, el cuál se agrega al programa de C# para poder utilizar los métodos de la clase escrita en Java. Para crear el archivo .jar, se utiliza el comando jar de java en el símbolo del sistema, como se muestra a continuación:

```
jar -cf OperacionesMatematicas.jar OperacionesMatematicas.class
```

La opción -c crea un archivo de almacenamiento.

-f especifica el nombre del archivo de almacenamiento.

Una vez que se genera el archivo .jar, se utiliza el comando ikvmc que es una utilidad de IKVM.NET, para generar el archivo .dll a partir del archivo .jar:

```
ikvmc -target:library OperacionesMatematicas.jar-out:OperacionesMatematicas.dll.
```

En la aplicación de C# se realizará una referencia a este archivo .dll, para reutilizar los métodos creados en la clase de Java.

La opción – target:library Construye una librería a partir del archivo .jar -out:<NombreArchivo>, que especifica el nombre que tendrá el archivo de salida [14].

6.2 Aplicación en C# de Microsoft

Ya generado el archivo .dll con el comando ikvmc, se agrega una referencia de este archivo, en la aplicación de C#. Una vez agregada la referencia es posible utilizar la clase OperacionesMatematicas creada en Java, en la aplicación de C#. De esta forma se reutiliza el código de Java y se implementa la interoperabilidad entre los lenguajes de programación de Java y C#.

Además se agrega una referencia al archivo IKVM.OpenJDK.core.dll que se encuentra en la carpeta IKVM, para agregar al proyecto las clases base de Java.

El código de la clase (librería) de Java es el siguiente:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using interoperabilidad;

namespace CalculadoraCsharp
{
    //Clase que implementa una calculadora simple
   public partial class frmCalculadora : Form
   {
      //Variables globales de la clase
  //Enumeración
         private enum Entrada { NINGUNA, DIGITO, OPERADOR, CE };
      private Entrada ultimaEntrada;
      private bool puntoDecimal;
      private char operador;
      private int numOperandos;
      private double operando1;
      private double operando2;

      //Método constructor de la clase
      public frmCalculadora()
      {
         InitializeComponent();
         //Iniciación de variables
         ultimaEntrada = Entrada.NINGUNA;
         puntoDecimal = false;
         operador = '0';
         numOperandos = 0;
         operando1 = 0;
         operando2 = 0;
      }

      //Método que controla el evento al dar clic a algún número
       private void btnNumeros_Click(object sender, EventArgs e)
      {
         //Obtiene el botón presionado
         Button btnN = (Button)sender;
        //Almacena el número del botón presionado como String
         String caracterBoton = btnN.Text;
             //Compara la entrada del botón anterior
          if (ultimaEntrada != Entrada.DIGITO)
           {
            if (caracterBoton.Equals("0"))
              return;
//Limpia el cuadro de texto donde se muestran los //números
            txtNumeros.Text="";
             //Almacena como ultima entrada un dígito
            ultimaEntrada = Entrada.DIGITO;
//El pundo decimal es falso para poder agregar un //punto decimal
            puntoDecimal = false;
          }
          //Muestra el número en el cuadro de texto
           txtNumeros.Text=txtNumeros.Text + btnN.Text;
      }
      //Método que se ejecuta cuando se presiona el botón punto
       private void btnPunto_Click(object sender, EventArgs e)
      {
        //Si la última entrada no fue un dígito
         if (ultimaEntrada != Entrada.DIGITO)
         {
            //Agrega un 0 antes del punto
            txtNumeros.Text="0.";
            ultimaEntrada = Entrada.DIGITO;
           //Indica que el punto decima ya fue presionado
            puntoDecimal = true;
         }
 //no permite presionar el punto dos veces
         else if (puntoDecimal == false)
         {
            //Muestra el punto decimal
```

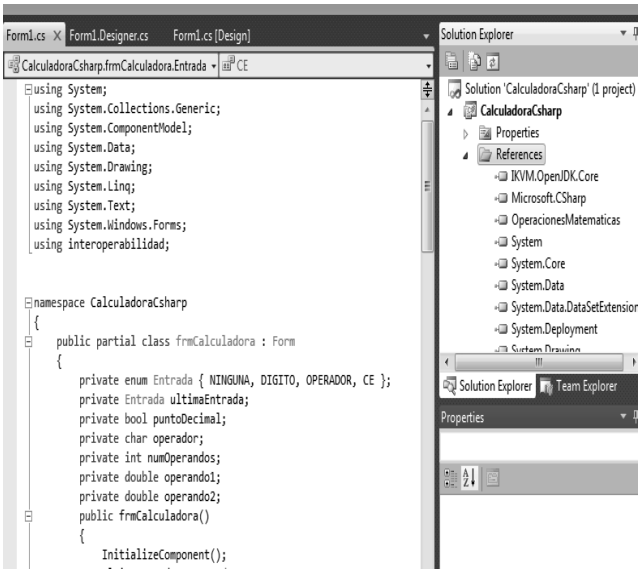
```
            txtNumeros.Text=txtNumeros.Text
+ ".";
          //Indica que el punto decima ya fue
presionado
            puntoDecimal = true;
        }
     }

//Método que se ejecuta cuando un botón
de algún operador  //es presionado

private void btnOperador_Click(object sen-
der, EventArgs e)
  {
//Crea el objeto op de la clase Operaciones-
Matemáticas                    //creada
en Java. Implementa la interoperabilidad en-
tre //lenguaje //de Java y C#
       OperacionesMatematicas      op=new
OperacionesMatematicas();
         //Guarda el botón presionado
         Button btnO = (Button)sender;
           //Almacena el operador del botón
presionado
         String operadorBoton = btnO.Text;
//Cuando no hay números capturados toma
el - como dígito para //indicar que es un nú-
mero negativo
       if ((numOperandos == 0) && operador-
Boton.Equals("-"))
     ultimaEntrada = Entrada.DIGITO;
          //Si el botón anterior presionada fue
un dígito
        if (ultimaEntrada == Entrada.DIGITO)
        //Incrementa el número de operandos
            numOperandos++;
           //Cuando el número de operandos
es 1
         if (numOperandos == 1)
           //El número del cuadro de texto el
guardado en operando1
       operando1 = Double.Parse(txtNumeros.
Text);
         //Si el número de operandos es 2 rea-
liza la operación
        else if (numOperandos == 2)
        {
           //Asigna a operando2 el valor del
cuadro de texto
      operando2 = Double.Parse(txtNumeros.
Text);
            switch (operador)
            {
     //Cuando el botón presionado es el mas
              case '+':
//Para realizar la suma utiliza el método de
//la clase de Java (interoperabilidad)
  op.setSuma(operando1, operando2);
//Obtiene el valor de la variable de //ins-
tancia suma de la clase de //Java
                operando1 = op.getSuma();
                break;
              case '-':
          op.setResta(operando1, operando2);
                operando1 = op.getResta();
                break;
              case 'X':
   op.setMultiplicacion(operando1,operando2);
          operando1 = op.getMultiplicacion();
                break;
              case '/':
       op.setDivision(operando1, operando2);
                operando1 = op.getDivision();
                break;
              case '=':
                operando1 = operando2;
                break;
              case '^':
       op.setPotencia(operando1, operando2);
                operando1 = op.getPotencia();
                break;
            }
      //Convierte el número decimal a String
                    txtNumeros.Text=Convert.
ToString(operando1);
           numOperandos = 1;
         }
          operador = operadorBoton[0];
       ultimaEntrada = Entrada.OPERADOR;
      }
```

```
    //Método que se ejecuta cuando se presiona el botón C
     private void btnC_Click(object sender, EventArgs e)
    {
      //Inicializa las variables
      txtNumeros.Text="0.";
      ultimaEntrada = Entrada.NINGUNA;
      puntoDecimal = false;
      operador = '0';
      numOperandos = 0;
      operando1 = 0;
      operando2 = 0;
    }
```

```
    //Método que se ejecuta cuando se presiona el botón CE
    private void btnCE_Click(object sender, EventArgs e)
    {
      //Borra la última entrada
      txtNumeros.Text="0.";
      ultimaEntrada = Entrada.CE;
       puntoDecimal = false;
    }
  }
} [15]
```

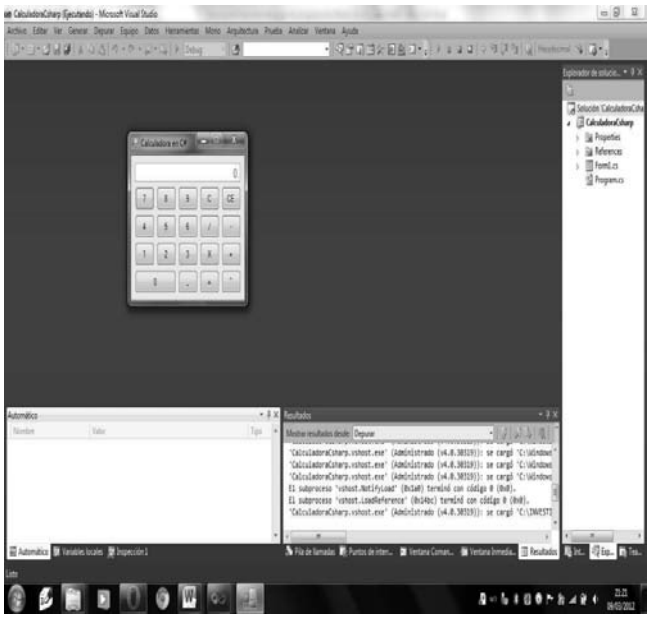
El código en visual studio se muestra en la figura 8.

Figura 8. Código en visual studio [15].



El resultado de esta aplicación se muestra en la figura 9.

Figura 9. Se muestra el uso de la calculadora [15].



7. 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 que herramientas o que 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 a nivel aparatos domésticos, no facilita del todo la interoperabilidad que es una nueva tendencia para el desarrollo de software.

Por lo que, la neutralidad del lenguaje ha incrementado mucho el atractivo de la tecnología .NET (pricipalmente c#), en especial cuando se considera el costo 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.

Sin embrago, en la actualidad, ya no es necesario adoptar otro lenguaje diferente al que usamos con la tecnología .NET, ya que incluso los paquetes de java se pueden convertir a código CIL utilizando una versión de java.Net, para que se trabaje desde visual studio; que es en lo que aterriza precisamente este trabajo.

Pero no solo desde visual studio se puede trabajar .Net, sino que también desde mono que es un entorno de desarrollo para .Net de software libre; en el cual también existe c#, java, mono basic, que es similar a visual basic de Microsoft, entre otros lenguajes.

En cualquier caso, Java y C# son dos lenguajes similares, sobre todo a nivel semántico, una capacidad que permite la existencia de aplicaciones que facilitan la conversión entre ambos lenguajes.

Sin embargo, en la actualidad, ya no es necesaria la conversión, al menos de java a otros lenguaje, ya que se puede convertir el código java a CIL para que pueda ser usado por otros lenguajes.

8. Referencias

[1] Schildt Herbert (2003), C#. Manual de referencia, Mc Graw Hill.

[2] Deitel P. J.; Deitel H. M, Java. Como Programar. Séptima Edición, Pearson Prentice Hall.

[3] León Rivera Said, Análisis comparativo de los principales paradigmas de programación, Unidad Académica de Ingeniería de la Universidad Autónoma de Guerrero.

[4] V. Aho, Alfred; et al, Compiladores. Principios, técnicas y herramientas, Pearson Addison Wesley.

[5] José Antonio González Seco, C# El nuevo lenguaje de Internet, http://mygnet.com, fecha de consulta 6 de mayo de 2011

[6] MSDN, (2011), Interoperabilidad entre lenguajes http://msdn.microsoft.com/es-es/library/a2c7tshk.aspx, fecha de consulta 15 de mayo del 2011.

[7] Fundación Josep Carreras, (2011)¿,Qué es .NET? http://globaliza.blogia.com/temas/tecnologia.net.php, Fecha de Consulta 14 de Mayo del 2011.

[8] Windu, Mace, (2011), http://www.portalhacker.net/index.php/topic,63240.0.html, Fecha de Consulta 16 de Mayo del 2011.

[9] adrformacion, (2011), http://www.adrformacion.com/curso/puntonet/leccion1/tecnologia_punto_net.htm, Fecha de Consulta 15 de Mayo del 2011

[10] Carrera Díaz Verónica, Características de la POO, Unidad Académica de Ingeniería de la Universidad Autónoma de Guerrero.

[11] Wikipedia, http://es.wikipedia.org/wiki/, fecha de consulta 3 de mayo de 20011.

[12] http://www.wilsonmar.com/msdotnet.htm, fecha de consulta 9 de enero de 2012.

[13] Severino Feliciano Morales, Interoperabilidad entre Lenguajes de Programación, Revista VÍNCULOS, Universidad Distrital Francisco José de Caldas, Bogotá Colombia. Pags. 186-195.

[14] Nicolás Parcerisa, Interoperabilidad entre java | .Net (2011), http://blog.hexacta.com/hat/interoperabilidad-java-net/, fecha de consulta 5 de marzo de 2012.

[15] Carrera Silva Humberto, Interoperabilidad entre lenguajes de programación, Unidad Académica de Ingeniería de la Universidad Autónoma de Guerrero, 2011.