CAPITULO III

HERRAMIENTAS DE DESARROLLO

CONTENIDO

3.1 Estudio de herramientas para el desarrollo de Servicios Web.

Microsoft Visual Studio.NET es un conjunto de múltiples lenguajes herramientas de programación para crear aplicaciones en la plataforma Microsoft .NET.

El .NET Framework representa el conjunto de interfaces de programación que forman el núcleo de la plataforma .NET y que permiten maximizar el rendimiento, la fiabilidad y la seguridad de los servicios Web XML.

Todos los lenguajes .NET emplean el Common Language Runtime (CLR) y comparten un gran conjunto de recursos como son: [WWW009]

- Un **modelo de programación** orientado a objetos (herencia, polimorfismo, manejo de excepciones y colección de basura).
- **Modelo de seguridad.**
- **Sistema de tipos.**
- **Base Class Library** (BCL) (Biblioteca de Clases Base).
- Desarrollo, depuramiento y herramientas.
- **Administración** de ejecución y código.
- Traductores y optimizadores de **MSIL** a código nativo.

3.1.1 C#

C# es el lenguaje de Microsoft para la plataforma .NET. Ha sido diseñado por Scott Wiltamuth y Anders Hejlsberg, éste último también diseñador del lenguaje Turbo Pascal y Delphi.

C# es el único que ha sido diseñado específicamente para ser utilizado en esta plataforma, por lo que programar usando C# es 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 y fácilmente legible que recoge las mejores características de muchos otros lenguajes, fundamentalmente Visual Basic, Java y C++, para combinarlas en uno sólo en el que se une la alta productividad y facilidad de aprendizaje de Visual Basic con la potencia de C++. [LIB009]:

- ***Características de C#***

  - Visual C# distingue entre minúsculas y mayúsculas.
  - Dispone de todas las características propias de cualquier lenguaje orientado a objetos: encapsulación, herencia y polimorfismo.
  - Tiene a su disposición un recolector de basura que libera al programador de la tarea de tener que eliminar las referencias a objetos que no son útiles, evitando que se agote la memoria en áreas ya liberadas y reasignadas.
  - Incluye soporte nativo para **EVENTOS Y DELEGADOS**. Los delegados son similares a los punteros a funciones y los eventos son mecanismos mediante los cuales los objetos pueden notificar de la ocurrencia de sucesos.
  - Incorpora propiedades, que permiten el acceso controlado a miembros de una clase. Gracias a ellos

se evita la pérdida de legibilidad que en otros lenguajes causa la utilización de métodos Set() y Get() y se mantienen todas las ventajas de un acceso controlado.

- Admite atributos que no son miembros de las clases sino información sobre éstas que podemos incluir en su declaración.

- Es un lenguaje que controla que todas las conversiones entre tipos se realicen de forma compatible.

- Asegura que nunca se acceda fuera del espacio de memoria ocupado por un objeto evitando frecuentes errores de programación y consigue que los programas no puedan poner en peligro la integridad de otras aplicaciones.

- Incluye la recolección de elementos no utilizados y la seguridad en el tratamiento de tipos.

- Seguridad implementada por medio de mecanismos de confianza intrínsecos del código.

- Compatible con componentes XML basados en Web y metadatos extensibles.

- Plena interoperabilidad por medio de los servicios de COM+ y .NET Framework con un acceso basado en bibliotecas permitiendo la integración de código existente.

- Proporciona interoperabilidad con otros lenguajes, entre plataformas y con datos heredados.

- Admite el control de versiones para facilitar la administración y la implementación.

Ejemplo de sintaxis en C#

```
Declarar y utilizar métodos

// Declaración de una funciòn de tipo void
void voidfunction() {
 ...
}

// Declara una función que retorna un valor
String stringfunction() {
 ...
   return (String) val;
}

// Declaración de una función que recibe y retorna valor
String parmfunction(String a, String b) {
 ...
   return (String) (a + b);
}

// Uso de la función
voidfunction();
String s1 = stringfunction();
String s2 = parmfunction("Hello", "World!");
```

Figura 3.1 ***Declarar y utilizar métodos mediante C#***

3.1.2 Visual Basic .NET

Visual Basic.NET ofrece un sin número de mejoras importantes sobre versiones anteriores de Visual Basic en lo que se refiere al soporte orientado a objetos, como es polimorfismo, herencia, sobrecarga de operadores y métodos. Es posible que libremente se pueda pasar tipos de datos desde y hacia componentes desarrollados en otros lenguajes, y heredar clases bases desarrolladas en otros lenguajes. Con Visual Basic .NET, los programadores pueden seguir aprovechando las capacidades de esta herramienta. [LIB010]

- ***Características de Visual Basic. Net***

- Es el descendiente de Visual Basic. Un programador se siente familiarizado inmediatamente con el lenguaje.
- Su sintaxis y semántica son simples, sencillas y fáciles de comprender. El lenguaje evita características poco intuitivas.
- Proporciona a los programadores las características principales de .NET Framework y es coherente con las convenciones del marco de trabajo.
- Es razonablemente actualizable partiendo de Visual Basic.
- Debido a que .NET Framework admite explícitamente varios lenguajes de programación, funciona bien en un entorno multilenguaje.
- Es compatible con versiones anteriores de Visual Basic. Visual Basic .NET tiene la misma sintaxis.
- Posee la misma semántica y el mismo comportamiento en tiempo de ejecución que sus predecesores.
- Es uno de los lenguajes más seguros en los que se puede programar. Visual Basic crea un equilibrio entre confiabilidad, facilidad de uso y eficiencia en la definición del lenguaje.
- Es un lenguaje extremadamente fácil de utilizar.
- Permite un desarrollo rápido del programa sin que se vea afectada la confiabilidad.
- Produce código predecible y eficiente.
- Funciona como un lenguaje de tipos con establecimiento flexible de tipos, para lograr un código de usuario más correcto y un desarrollo más rápido en el último.
- Visual. Net no distingue entre minúsculas y mayúsculas.

Ejemplo de sintaxis en C#

```
´ Declaración de una función de tipo void
Sub VoidFunction()
 ...
End Sub

´Declara una función que retorna un valor
Function StringFunction() As String
 ...
   Return CStr(val)
End Function

'  Declaración de una función que recibe y retorna valor
Function ParmFunction(a As String, b As String) As String
 ...
   Return CStr(A & B)
End Function

' Uso de la función
VoidFunction()
Dim s1 As String = StringFunction()
Dim s2 As String = ParmFunction("Hello", "World!")
```

Figura 3.2 ***Declarar y utilizar métodos mediante Visual Basic.Net***

3.1.3 C++.Net

La plataforma .NET también utiliza el lenguaje Microsoft Visual C++. Sin embargo, al ser un lenguaje administrado, no puede ser manejado directamente por CLR, por ello, Microsoft ha agregado un conjunto de ***EXTENSIONES ADMINISTRADAS*** para Visual C++. El código escrito con estas extensiones cumple con el CLS, logrando desarrollar la ***compatibilidad de C++ con la plataforma .NET***, y bajo el control del .NET Framework, mientras que las clases C++ tradicionales no administradas siguen ejecutándose en el ambiente basado en Microsoft Windows.[LIB003]

Las ***extensiones administradas*** son nuevas palabras reservadas y atributos en el sistema de desarrollo Visual C++, que *permiten decidir que clases y funciones compilar como código administrado o no administrado.* Estas clases podrán posteriormente interoperar entre si y con bibliotecas externas.

Las extensiones administradas para C++ se crearon para ampliar el lenguaje C++ y permitir al usuario utilizar .NET Framework y orientar Common Language Runtime sin necesidad de aprender un nuevo lenguaje de programación.

Las extensiones administradas son útiles si:

- Se requiere migrar en etapas una gran porción de código, desde C++ no administrado hacia la plataforma .NET
- Si en una aplicación se tiene componentes C++ no administrados que se deseen utilizar desde las aplicaciones .NET Framework
- Si se tiene componentes .NET Framework que desee utilizar desde C++ no administrado.
- Cuando se desea mezclar código C++ no administrado y código .NET en la misma aplicación.

• Características de las extensiones administradas en C++

- ***Suave migración de código existente a .NET***

  Las extensiones administradas para C++ ofrecen flexibilidad para los desarrolladores que se enfocan en la plataforma .NET.

- ***Accediendo a clases .NET desde código nativo***

  El código C++ tradicional no administrado y C++

administrado se pueden mezclar libremente dentro de la misma aplicación.

- ***Código administrado y nativo en un mismo ejecutable*** Las nuevas aplicaciones escritas con extensiones administradas pueden aprovechar lo mejor de ambos mundos.
- ***Acceder a un componente C++ desde un lenguaje .NET*** Los componentes existentes pueden empaquetarse fácilmente como componentes .NET utilizando las extensiones administradas, conservando la inversión en el código existente al tiempo que se integra con .NET.
- ***El código administrado ofrece mayor productividad al desarrollador*** debido a las características tales como colección de basura y bibliotecas de clase.

Ejemplo de sintaxis en C++

```
// Declaración de una función de tipo void
void voidfunction() {
 ...
}

// Declara una función que retorna un valor
String stringfunction() {
 ...
   return (String) val;
}

// Declaración de una función que recibe y retorna valor
String parmfunction(String a, String b) {
 ...
   return (String) (a + b);
}

// Uso de la función
voidfunction();
String s1 = stringfunction();
String s2 = parmfunction("Hello", "World!");
```

Figura 3.3 ***Declarar y utilizar métodosmediante C++.Net***

3.1.4 J#

Visual J# está orientado a Common Language Runtime y se puede utilizar para desarrollar aplicaciones .NET, incluidos Servicios Web XML y aplicaciones Web, de forma que se hace un uso total de .NET Framework. [WWW011]

- ***Características de J#***

  - El compilador de Visual J#, compila archivos de código fuente de Java como Lenguaje intermedio de Microsoft® (MSIL).
  - Posee un conversor binario que transforma código de bytes de Java en Lenguaje intermedio de Microsoft (MSIL).
  - Las bibliotecas de clases son desarrolladas de manera independiente y diseñadas para proporcionar la funcionalidad de la mayoría de las bibliotecas de clases de JDK
  - Compilador de Visual J# (vjc.exe)
  - Conversor binario de Visual J# (JbImp.exe)
  - Un conjunto de biblioteca de clases desarrolladas de manera independiente y diseñada para proporcionar la funcionalidad de la mayoría de las bibliotecas de clases.
  - Las bibliotecas de clases que se distribuyen con Visual J# son capas basadas en .NET Framework y Common Language Runtime.
  - Integración entre lenguajes.

- Seguridad mejorada.
- Control de versiones e implementación.
- Servicios de depuración y generación de perfiles

Se puede utilizar Visual Studio para depurar aplicaciones de Java, incluso si no se tiene Visual J# instalado en el equipo. La extensión predeterminada de archivo de código fuente en Visual J# es .jsl.

Ejemplo de sintaxis en J#

```
// Declaración de una función de tipo void
function voidfunction() : void {
  ...
}

// Declara una función que retorna un valor
     function stringfunction() : String {
      ...
       return String(val);
}

// Declaración de una función que recibe y retorna valor
function parmfunction(a:String, b:String) : String {
      ...
 return String(a + b);
}

// Uso de la función
   voidfunction();
   var s1:String = stringfunction();
   var s2:String = parmfunction("Hello", "World!");
```

Figura 3.4 ***Declarar y utilizar método smediante J#***

3.2 Compatibilidad de herramientas integradas a .Net.

En programación se utiliza una gran variedad de herramientas y tecnologías, cada una de las cuales podría admitir distintos tipos y características, pero siempre ha sido complicado garantizar la *interoperabilidad entre lenguajes*.

La compatibilidad 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**. [LIB003]

Common Language Runtime ofrece la base para la interoperabilidad entre lenguajes al especificar e imponer tres componentes principales:

a) Un sistema de tipos, define cómo se declaran, utilizan y administran los tipos en el motor de tiempo de ejecución. El sistema de tipos permite la integración de lenguajes mediante los siguientes aspectos:

- Seguridad de tipos y la ejecución de código con alto rendimiento entre lenguajes.
- Proporciona un modelo orientado a objetos que admite la implementación completa de muchos lenguajes de programación.
- Define reglas que deben seguir los lenguajes, garantizando que los objetos escritos en distintos lenguajes puedan interactuar unos con otros.

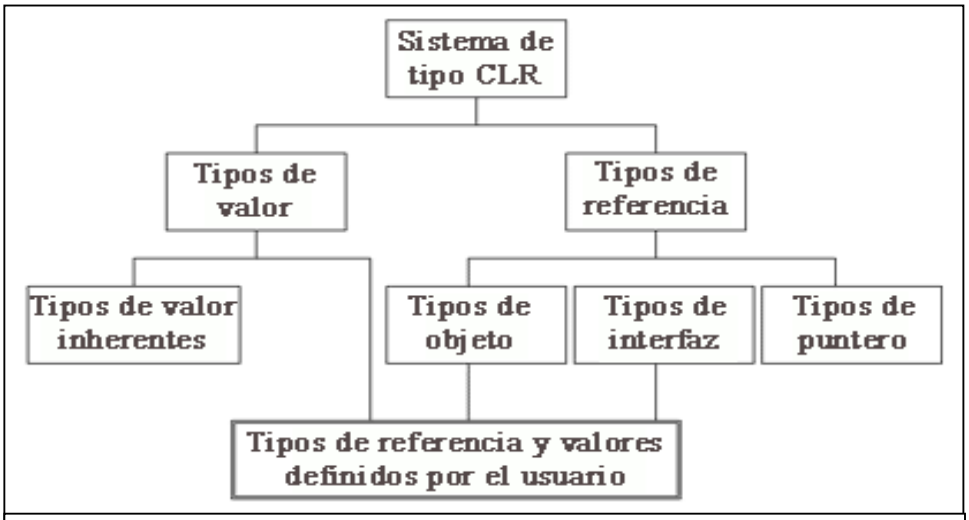

Figura 3.5 ***Estructura del Sistema de Tipos Común CTS de .NET***

b) Un sistema de metadatos, mediante la definición de un mecanismo para almacenar y recuperar la información de CTS. Los compiladores almacenan la información sobre tipos como metadatos y Common Language Runtime usa esta información para proporcionar servicios durante la ejecución; el motor de tiempo de ejecución puede administrar la ejecución de aplicaciones de múltiples lenguajes porque toda la información de tipos se almacena y recupera de la misma forma, independientemente del lenguaje en que se haya escrito el código. Los compiladores para .NET Framework describen los tipos que producen con metadatos por dos motivos:

- Los metadatos permiten que los tipos definidos en un lenguaje puedan utilizarse en otro lenguaje.
- El motor de ejecución requiere que los metadatos administren objetos. y esta acción incluye requisitos como la administración de memoria.

c) Si el CTS es la unión de distintas características de los lenguajes de programación, la especificación en lenguaje común (CLS) constituye las normas que han de cumplir estas características comunes de los lenguajes de programación.

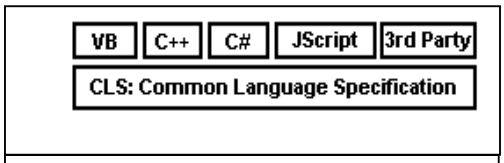

Figura 3.6 ***Estructura de CLS***

El CLS representa un nivel de compatibilidad que la mayoría de los lenguajes deberán ser capaces de conseguir si los compiladores desean interoperabilidad.

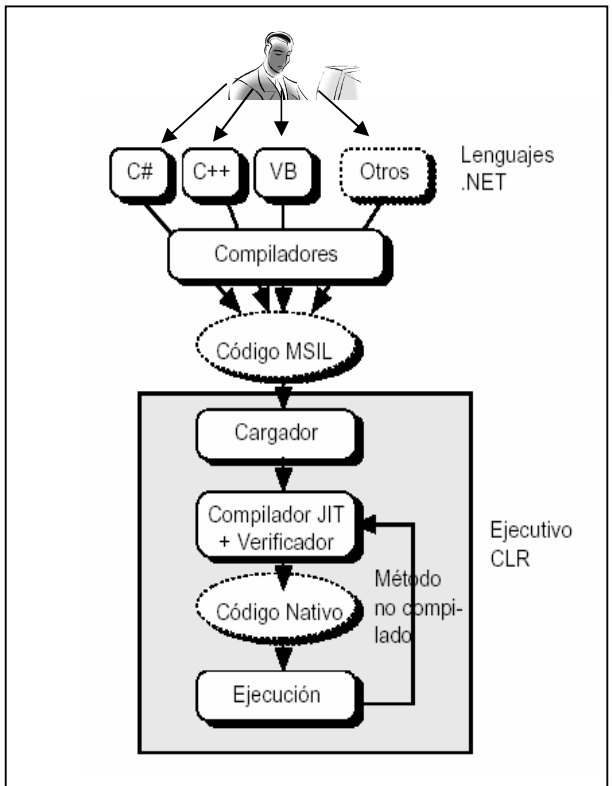

Figura 3.7 ***Proceso de compilación de una aplicación***

En la figura 3.7 se muestra la relación entre los distintos elementos del tiempo de ejecución durante el proceso de compilación de una aplicación.

a) El programador escribe el código fuente del programa usando el lenguaje .NET que prefiera(C#, Visual Basic, J# o C++ administrado, etc.)

b) El código fuente es compilado usando el compilador apropiado:

C# usa CSC.EXE
Visual Basic.Net usa VBC.EXE
J# usa VJC.EXE
Jscript.NET usa JSC.EXE

c) El compilador convierte el código fuente en un lenguaje intermedio MSIL.

1.- No siempre código no administrado

2.- El resultado se guarda bajo la forma de una DLL o un EXE denominados ensamblados y solo se puede usar / ejecutar en un sistema que disponga de la plataforma .NET

d) El fichero compilado es "interpretado" por un intérprete de MSIL.

- Primero se comprueba el código para seguridad en los tipos de datos.

- Después se activa el JUST In Time (JIT) que compila el MSIL en código nativo administrado y se combina con el Common Lenguaje Runtime (CLR) para producir el resultado final y se almacena en un archivo junto con los metadatos del nuevo tipo. El formato de metadatos utilizado es independiente del lenguaje de programación en el que se definió primeramente el tipo y es accesible a cualquier lenguaje.

3.3 Ventajas y desventajas de integración de código en .Net

Microsoft .NET ofrece soporte oficial para Visual Basic.NET, C++.NET y C#. Otros lenguajes desarrollados por terceros están ya disponibles como COBOL, Eiffel o Delphi, pero .NET va más allá de soportar estos lenguajes sino que además ofrece interoperabilidad entre ellos, por lo que es posible construir un componente en un lenguaje, introducirlo en una aplicación escrita en otro distinto e incluso heredarlo y añadir nuevas características en un tercero.

Respecto a esta capacidad de Microsoft .NET de trabajar con varios lenguajes existen ventajas y desventajas:

- ***Ventajas de integración de código . Net***

  - **Permite una migración más sencilla para antiguos programadores**, reduciendo el tiempo de formación. El trabajar con un lenguaje conocido proporciona gran productividad individual.
  - **Las clases y estructuras se hallan centralizadas**, por lo que es posible realizar aplicaciones desde cualquier lenguaje de la plataforma .Net empleando la misma sintaxis de evocación.
  - **El resultado final es más consistente** ya que Microsoft no necesita mantener diferentes implementaciones de la misma funcionalidad.
  - **Independencia de las aplicaciones** del lenguaje de programación utilizado y desarrollo de aplicaciones multi-lenguaje.
  - Preserva inversión del desarrollador.

- ***Desventajas de integración de código . Net***

La existencia de varios lenguajes de programación en una única empresa acarrea efectos negativos:

- **La sencillez de mantenimiento se reduce**. Si una aplicación está realizada en varios lenguajes se necesitan expertos en varios lenguajes para entenderla y mantenerla, aumentando los costes considerablemente.
- **La productividad del grupo decrece**. Si los programadores utilizan lenguajes diferentes no pueden comunicar fácilmente sus conocimientos de unos a otros.
- **Transferencia de conocimientos**. En el caso de que un desarrollador o grupo de desarrolladores especializados en un lenguaje dejan un proyecto, es necesario otros que conozcan el mismo lenguaje para continuar con el desarrollo.

En una empresa dedicada al desarrollo de software se debe seguir un criterio homogéneo y realizar todos sus desarrollos utilizando un único lenguaje, ya sea Java, C# o cualquier otro, independientemente de la plataforma utilizada.