ESQUEMA PARA LA PRESENTACIÓN DE PFC

1. INTRODUCCIÓN

Existen muchos serializadores en el mercado, cada uno con sus ventajas y desventajas, válidos para determinados tipos de objetos y no para otros, que generan código serializado para ser almacenado y/o transportado en diversos formatos y contextos.

Con este proyecto hemos conseguido, tras analizar una amplia variedad de serializadores, obtener el *HiperSerializador*, un serializador versátil que sirva para cualquier tipo de objeto, generando cualquier tipo de código serializado transportable y almacenable en cualquier contexto, capaz de reconocer los atributos estándar o cualquier atributo particular que se desee generar, ampliable mediante plug-ins y por supuesto, con los mejores resultados en cuanto a rendimiento.

Esta aplicación es un **generador de serializadores particulares de alto rendimiento**.

2. SERIALIZACIÓN Y SERIALIZADORES

2.1. Serializadores existentes

2.2. Factores a tener en cuenta

2.3. Comparativa de características

3. HIPERSERIALIZADOR

3.1. Qué es HiperSerializer

Es un pequeño programa que tiene una gran potencia.

3.2. Función de HiperSerializer

Generar un serializador para cualquier clase cuyo tipo se le indique en el constructor. Este serializador realiza la tarea de serializar y deserializar cualquier objeto de esa clase de manera óptima.

3.3. Cómo funciona el programa

El programa recibe como entrada un tipo de dato de .Net, realiza una comprobación de todos los elementos susceptibles de ser serializados, y va generando el código de una aplicación capaz de realizar la serialización de instancias de ese tipo. A continuación, compila al vuelo ese código, instancia un objeto de ese programa y eso es lo que devuelve como salida.

El programa generado realiza la tarea exclusiva de serializar y deserializar objetos de ese tipo. Además, lo hace de una manera muy eficiente y rápida, como se verá en las pruebas realizadas.

La aplicación Generador.dll, se integra en cualquier aplicación .net simplemente cargándose como un DLL más.

A partir de ese momento, se puede utilizar instanciando la clase Generador y llamando al método getSerializer. A la hora de instanciar, al constructor hay que pasarle como parámetro un System.Type. En el momento de invocar al método getSerializer, se obtendrá el serializador particular para ese tipo en una variable de tipo dinámico, ya que el objeto devuelto será de una clase particular no disponible en tiempo de compilación.

3.4. Cómo genera cada serializador

Una vez que se invoca el método Generador.getSerializer, el programa ya sabe para qué tipo tiene que generar el código de su serializador particular. Para ayudar en esta tarea se utiliza el espacio de nombres System.Reflection.

Reflection es una característica de .Net que permite trabajar con los elementos que componen un tipo, campos, propiedades, métodos, eventos, etc. Nos permite obtener su nombre, tipo, y muchas más cualidades (si es público o privado, sus atributos, etc.).

Se usa Reflection para capturar todos los elementos del tipo susceptibles de ser serializados. Para cada uno de ellos, se realiza la tarea de generar el código que permite su serialización y deserialización e ir metiéndolo en un stream.

Para cada elemento, se generará el código que permitirá colocar en un string la información de su valor (serialización) o capturar de dicho string su valor para colocarlo en el elemento correspondiente (deserialización) según el formato que se desee para la salida de la serialización. Ambos procesos se hacen de manera sucesiva para cada elemento, por lo que el código de la serialización se genera a la vez que el código de la deserialización.

Por ejemplo: Clase01Basica

public class Clase01Basica

{

public string v1 = "1";

public int v2 = 2;

}

El código del serializador para esta clase se conformará así:

public class Clase01BasicaCodec{

public static string encode(Fase02.Clase01Basica obj){  
 StringBuilder texto = new StringBuilder();  
 texto.Append(obj.v1.ToString() + ",");  
 texto.Append(obj.v2.ToString() + ",");  
 return texto.ToString();  
 }

public static void decode(ref String codigo, ref Fase02.Clase01Basica obj){  
 Queue<string> elementos = new Queue<string>(codigo.Split(','));  
 obj.v1 = elementos.Dequeue();  
 obj.v2 = Int32.Parse(elementos.Dequeue());  
 }

}

Para cada elemento recogido con Reflection con getMembers() se añade la línea texto.Append(obj.v1.ToString() + “,”)

4. COMPARATIVA

5. MEJORAS

6. CONCLUSIONES